University Cote d'Azur

Polytech Nice Sophia

Department
Engineering of Electronic Systems

Labs
Real Time Operating System

Fabrice MULLER
🖂 : Fabrice.Muller@univ-cotedazur.fr

- 2020/2021 -

Contents

Part I

Scheduling Theory

LAB N° 1

Scheduling

1.1 Scheduling Policies

We consider 5 tasks T_1, T_2, T_3, T_4, T_5 whose Execution Times (ET) are as follows : $ET(T_1) = 10$, $ET(T_2) = 1$, $ET(T_3) = 2$, $ET(T_4) = 1$, $ET(T_5) = 5$. Give the scheduling corresponding to the following strategies. Give for each scheduling the Average Waiting Time AWT. In cases with interruption (preemption), give two calculations : a calculation where the context switch time is 0, and another where the context switch time takes 0.1 time unit. The arrival times (AT) is 0 except if they are specified.

1. FCFS (First Come First Serve), the order of arrival being T_1, T_2, T_3, T_4, T_5 (cf figure 1.1).

2. SJF (Shortest Job First), (cf figure 1.2).

3. Scheduling with priority without preemption (Non Preemptive Priority scheduling), the priorities being as follows : $P(T_1) = 2$, $P(T_2) = 4$, $P(T_3) = 2$, $P(T_4) = 1$, $P(T_5) = 3$. A higher priority corresponds to a greater number (0 = lower priority). Always use FCFS for arrival order (cf figure 1.3).

4. RR (Round Robin). Always use FCFS for arrival order.

   (a) With slot time = 1 and consider the following set of tasks that arrive at time 0ms. Complete the trace in the figure 1.4 and give the Average waiting time (AWT).

   (b) With slot time = 1 and consider now the following tasks T_1,T_2,T_3,T_4,T_5 arrive respectively at time 0ms, 1ms, 2ms, 3ms and 4ms. What is the Average waiting time with context switch time = 0 unit ? Conclusion ?

   (c) With slot time = 2 and consider the following set of tasks that arrive at time 0ms. Complete the trace in the figure 1.5 and give the Average waiting time (AWT).

   (d) With slot time = 2 and consider now the following tasks T_1,T_2,T_3,T_4,T_5 arrive respectively at time 0ms, 1ms, 2ms, 3ms and 4ms. What is the Average waiting time with context switch time = 0 unit ? Conclusion ?

(e) What is the conclusion about the time slot (quantum) and the waiting time ?

5. SRTF (Shortest Remaining Time First), the arrival times (AT) of the tasks being the following : $AT(T_1) = 0$, $AT(T_2) = 1$, $AT(T_3) = 2$, $AT(T_4) = 3$, $AT(T_5) = 4$ (cf figure 1.6).

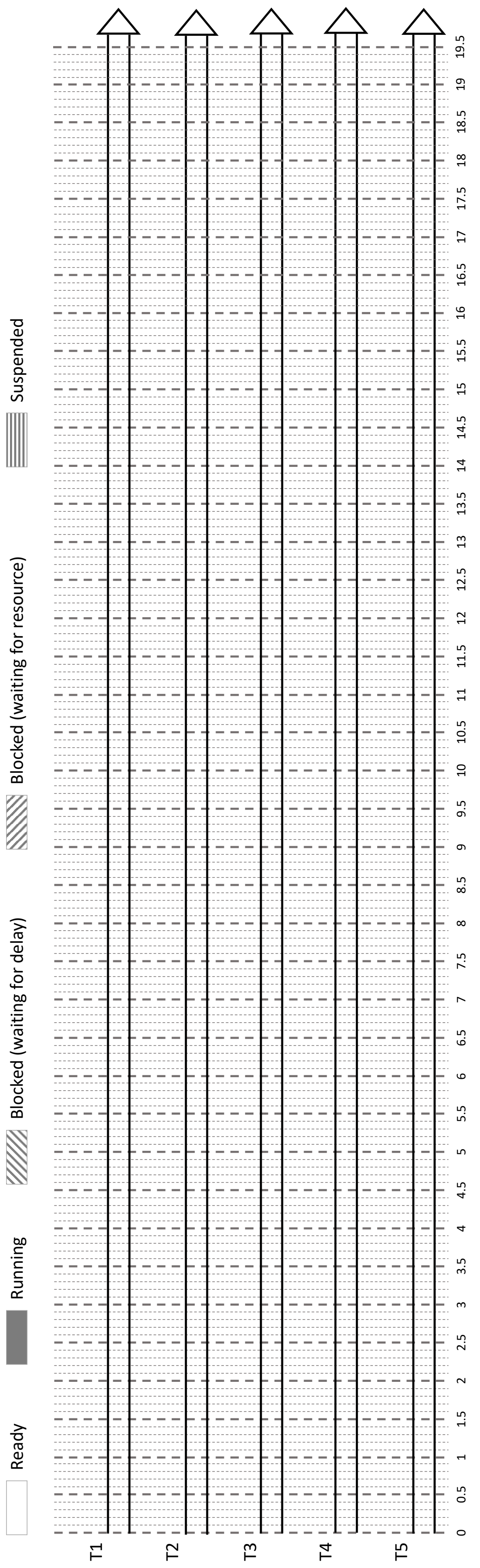


Figure 1.1 – *FCFS (First Come First Serve) trace.*

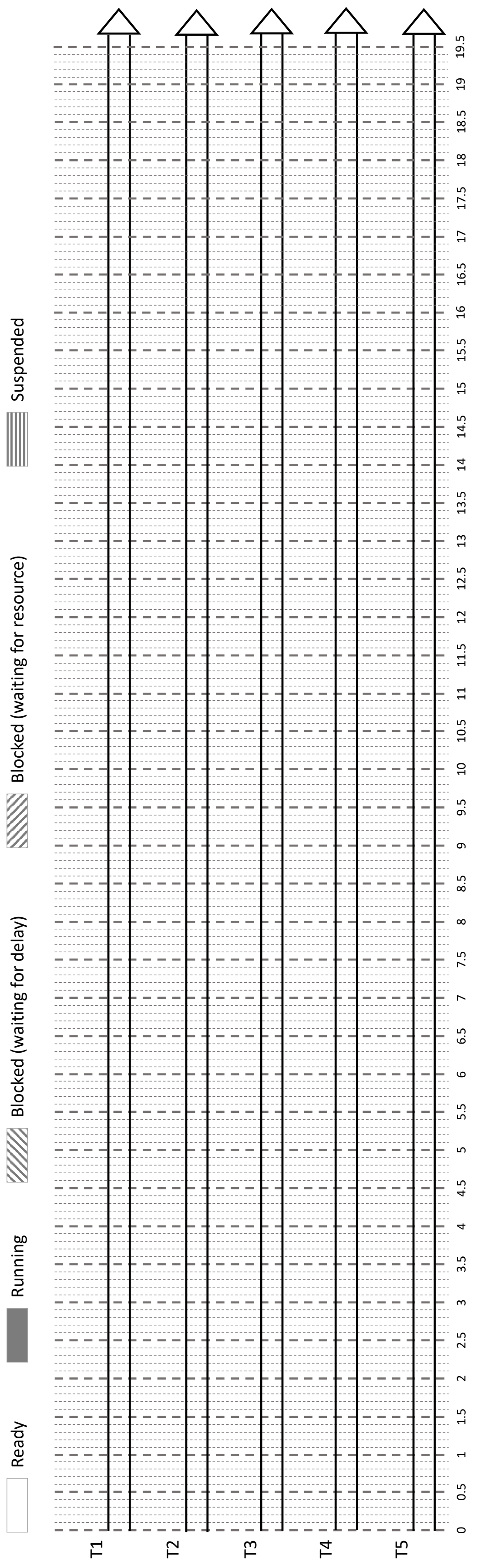


Figure 1.2 – *SJF (Shortest Job First) trace.*

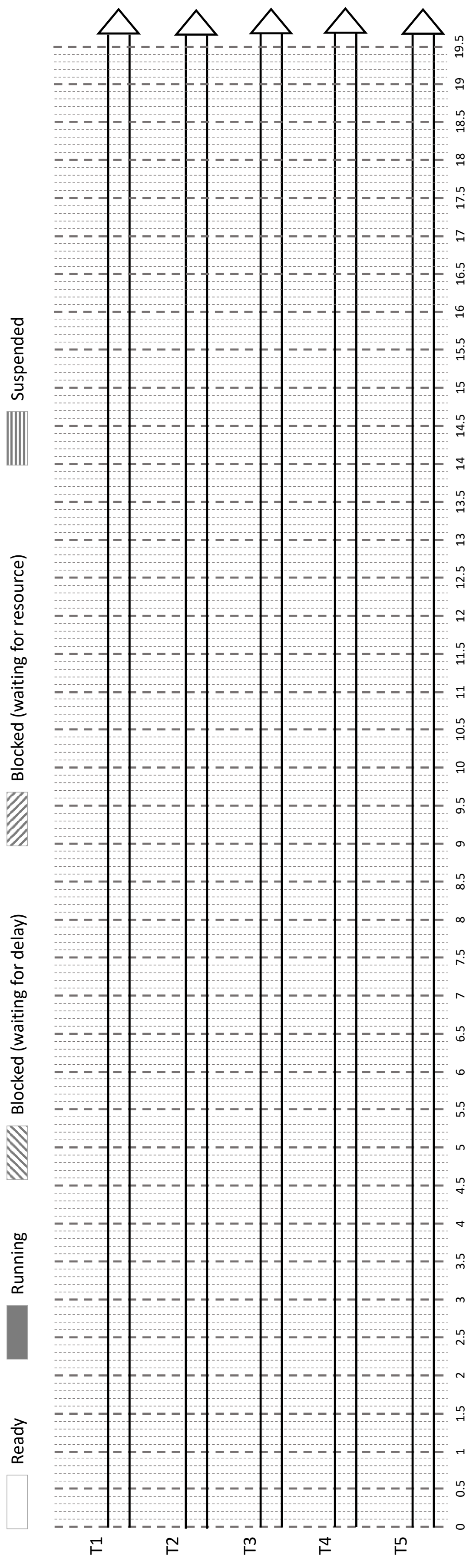


Figure 1.3 – *Non Preemptive Priority scheduling trace.*

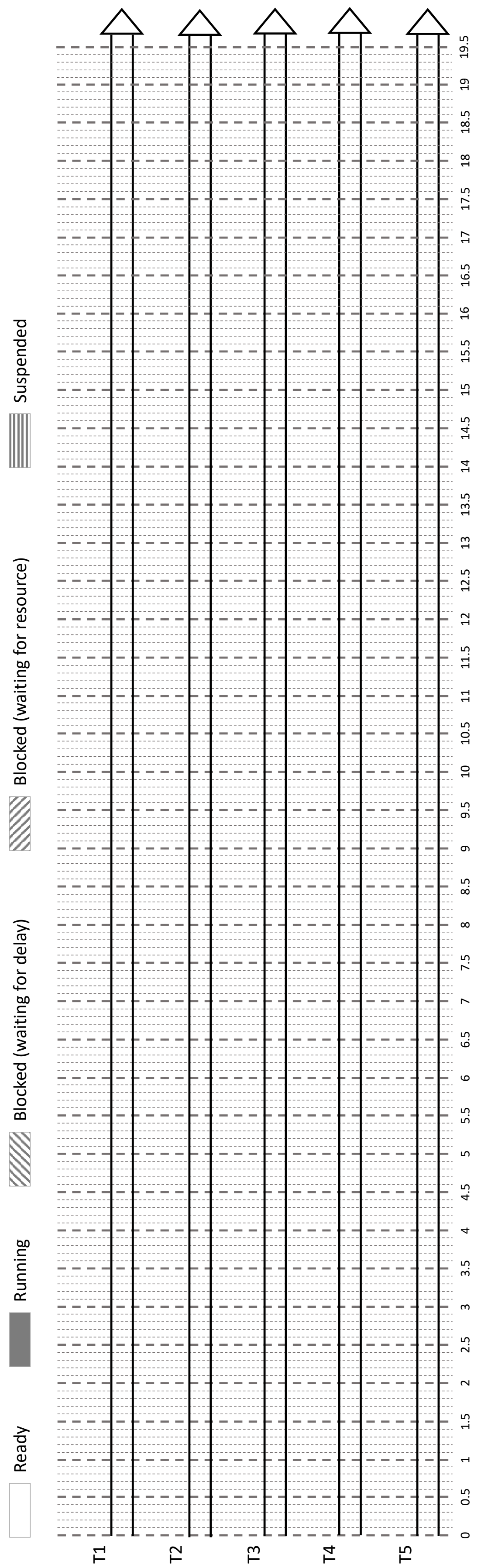


Figure 1.4 – *Round Robin (RR) scheduling trace with slot times = 1.*

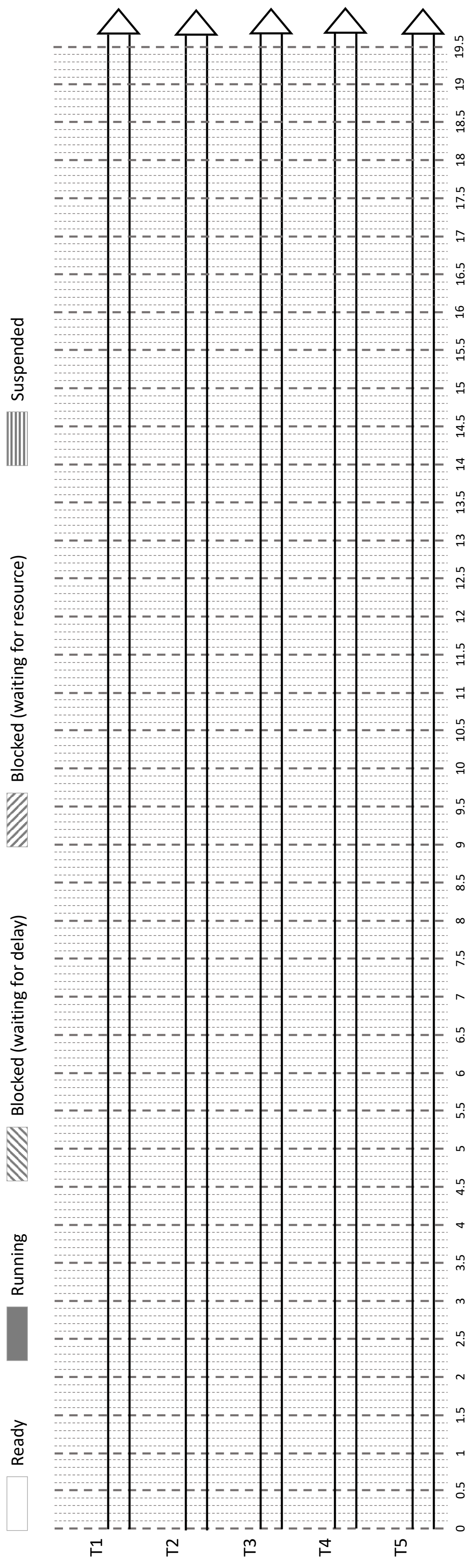


Figure 1.5 – *Round Robin (RR) scheduling trace with slot times = 2.*

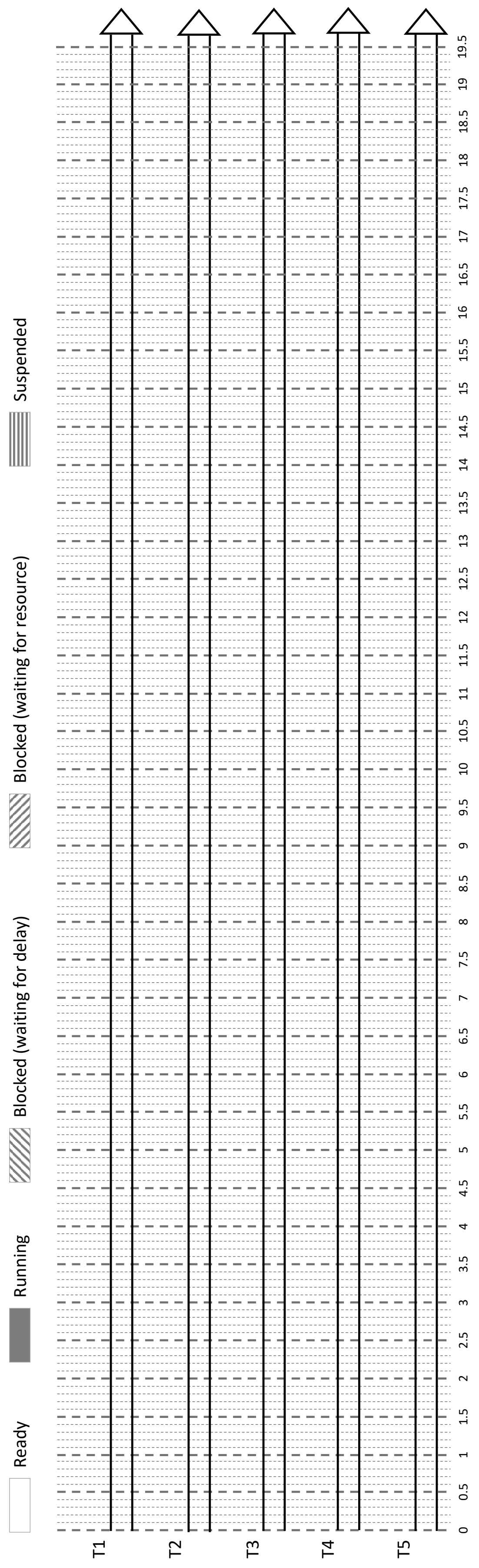


Figure 1.6 – *SRTF (Shortest Remaining Time First) scheduling trace.*

1.2 Schedulability

We consider 3 periodic tasks T_1, T_2 and T_3 whose characteristics are indicated in the table 1.1. The period is equal to the deadline and the tasks are independent.

Task	Execution Time (ET)	Period (P)
T_1	10	50
T_2	9	40
T_3	7	30

Table 1.1 – *Characteristics of tasks in milliseconds.*

1. Show that these tasks can be scheduled according to the Rate Monotonic Scheduling (RMS) strategy.

2. We now change the Execution Time of the task $T_3 = 11$. Is this always schedulable according to the RMS ?

3. Is this schedulable according to the EDF (Earliest Deadline First) scheduling for the previous question ?

4. What is the least common multiple of the period in milliseconds for EDF scheduling ?

Part II

Espressif Framework

LAB N° 1

Framework

Lab Objectives

- Understand the Espressif IoT Development Framework.
- Run a first program.
- Create an GitHub repository.
- Use the Microsoft Visual Studio Code with a dedicated ESP32 project template.

1.1 Espressif IoT Development Framework

We will start by understanding the structure of the Espressif IDF framework using an example provided by Espressif.

1.1.1 First look of the first example

Take the example « hello_world » which displays the string « hello world ! » and characteristics of the ESP32 board on the console. The example is located in the following directory :

```
esp32:~$ cp -R ~/esp/esp-idf/examples/get-started/hello_world hello_world
esp32:~$ cd hello_world
```

The compilation is done from a Python script called *idf.py*. This script is located in `~/esp/esp-idf/tools/` and added in the path.

```
esp32:~/esp/esp-idf/examples/get-started/hello_world$ which idf.py
/home/esp32/esp/esp-idf/tools/idf.py
```

```
esp32:~/esp/esp-idf/examples/get-started/hello_world$ env | grep esp-idf
IDF_TOOLS_EXPORT_CMD=/home/esp32/esp/esp-idf/export.sh
PWD=/home/esp32/esp/esp-idf/examples/get-started/hello_world
IDF_TOOLS_INSTALL_CMD=/home/esp32/esp/esp-idf/install.sh
IDF_PATH=/home/esp32/esp/esp-idf
PATH=/home/esp32/esp/esp-idf/components/esptool_py/esptool:/home/esp32/esp/esp-idf/
    components/espcoredump:/home/esp32/esp/esp-idf/components/partition_table/:/
    home/esp32/.espressif/tools/xtensa-esp32-elf/esp-2019r2-8.2.0/xtensa-esp32-elf/
    bin:/home/esp32/.espressif/tools/esp32ulp-elf/2.28.51.20170517/esp32ulp-elf-
    binutils/bin:/home/esp32/.espressif/tools/openocd-esp32/v0.10.0-esp32-20190313/
    openocd-esp32/bin:/home/esp32/.espressif/python_env/idf4.0_py3.6_env/bin:/home/
    esp32/esp/esp-idf/tools:/home/esp32/.local/bin:/usr/local/sbin:/usr/local/bin:/
    usr/sbin:/usr/bin:/sbin:/bin:/usr/games:/usr/local/games:/snap/bin
```

To display the help of the Python script, just type the name of the command as below. We will mainly use the following commands : `build, flash, monitor, menuconfig, fullclean, size ...`

```
esp32:~/esp/esp-idf/examples/get-started/hello_world$ idf.py
Checking Python dependencies...
Python requirements from /home/esp32/esp/esp-idf/requirements.txt are satisfied.
Usage: /home/esp32/esp/esp-idf/tools/idf.py [OPTIONS] COMMAND1 [ARGS]... [COMMAND2
    [ARGS]...]...

  ESP-IDF build management

Options:

  -b, --baud INTEGER              Baud rate. This option can be used at most once
    either globally, or
                                  for one subcommand.
...
  -p, --port TEXT                 Serial port. This option can be used at most once
     either globally,
                                  or for one subcommand.
...
Commands:
  all                    Aliases: build. Build the project.
...
  clean                  Delete build output files from the build directory.
...
  flash                  Flash the project.
  fullclean              Delete the entire build directory contents.
  menuconfig             Run "menuconfig" project configuration tool.
  monitor                Display serial output.
...
```

```
  size                    Print basic size information about the app.
  size-files              Print per-source-file size information.
```

The C source files are usually located in the « main » folder. We see below the « hello_world_main.c » file. The other files will be studied later.

```
esp32:~/esp/esp-idf/examples/get-started/hello_world$ cd main
esp32:~/esp/esp-idf/examples/get-started/hello_world$ ll

total 20
drwxr-xr-x 2 esp32 esp32 4096 avril  2 15:31 ./
drwxr-xr-x 4 esp32 esp32 4096 mai   26 10:07 ../
-rw-r--r-- 1 esp32 esp32   85 avril  2 15:31 CMakeLists.txt
-rw-r--r-- 1 esp32 esp32  146 avril  2 15:31 component.mk
-rw-r--r-- 1 esp32 esp32 1232 avril  2 15:31 hello_world_main.c
```

1.1.2 Building the first example

The generation of the executable in this specific case is called **cross-compilation** because the program will not be performed on the computer but on the ESP32 board. We build the executable from the following command.

```
esp32:~/esp/esp-idf/examples/get-started/hello_world$ idf.py build

[59/62] Linking C static library esp-idf/spi_flash/libspi_flash.a
[60/62] Linking C static library esp-idf/main/libmain.a
[61/62] Linking C executable bootloader.elf
[62/62] Generating binary image from built executable
esptool.py v2.8
Generated /home/esp32/esp/esp-idf/examples/get-started/hello_world/build/bootloader
    /bootloader.bin
[820/820] Generating binary image from built executable
esptool.py v2.8
Generated /home/esp32/esp/esp-idf/examples/get-started/hello_world/build/hello-
    world.bin

Project build complete. To flash, run this command:
/home/esp32/.espressif/python_env/idf4.0_py3.6_env/bin/python ../../../components/
    esptool_py/esptool/esptool.py -p (PORT) -b 460800 --before default_reset --
    after hard_reset write_flash --flash_mode dio --flash_size detect --flash_freq
    40m 0x1000 build/bootloader/bootloader.bin 0x8000 build/partition_table/
    partition-table.bin 0x10000 build/hello-world.bin
or run 'idf.py -p (PORT) flash'
```

A new « build » folder appears. In this folder, you can see the « hello-world.elf » which will be flashed in the ESP32 board.

```
esp32:~/esp/esp-idf/examples/get-started/hello_world$ ll
total 56
drwxr-xr-x  4 esp32 esp32  4096 mai   26 10:07 ./
drwxr-xr-x  4 esp32 esp32  4096 avril  2 15:31 ../
drwxr-xr-x 74 esp32 esp32  4096 mai   26 10:27 build/
-rw-r--r--  1 esp32 esp32   234 avril  2 15:31 CMakeLists.txt
drwxr-xr-x  2 esp32 esp32  4096 avril  2 15:31 main/
-rw-r--r--  1 esp32 esp32   183 avril  2 15:31 Makefile
-rw-r--r--  1 esp32 esp32   170 avril  2 15:31 README.md
-rw-r--r--  1 esp32 esp32 25463 mai   26 10:26 sdkconfig

esp32:~/esp/esp-idf/examples/get-started/hello_world$ cd build

esp32:~/esp/esp-idf/examples/get-started/hello_world/build$ ll hello-world*
-rw-r--r-- 1 esp32 esp32  147232 mai   26 10:27 hello-world.bin
-rwxr-xr-x 1 esp32 esp32 2451528 mai   26 10:27 hello-world.elf*
-rw-r--r-- 1 esp32 esp32 1541555 mai   26 10:27 hello-world.map

esp32:~/esp/esp-idf/examples/get-started/hello_world/build$ cd ..
```

1.1.3 Running the first example on ESP32 board

You find details of the ESP32-PICO-KIT board in the Getting Started Guide. To run the program on the board, follow the procedure below :

- Connect the ESP32 card to the computer via USB (cf. figure 1.1)

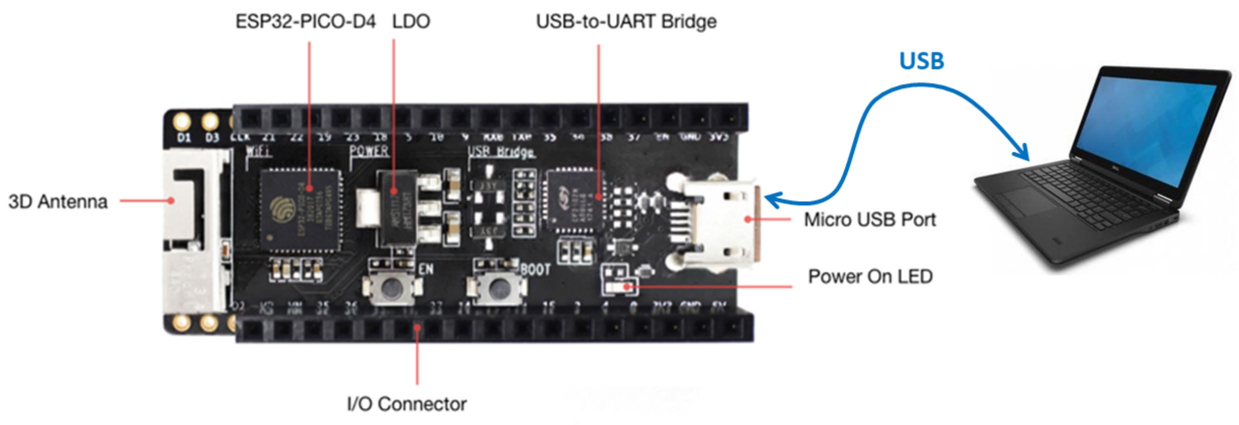


Figure 1.1 – *ESP32-PICO-KIT board connections.*

- Identify the USB serial port (usually /dev/ttyUSB0)

```
esp32:~/esp/esp-idf/examples/get-started/hello_world$ ls /dev/ttyUSB*
/dev/ttyUSB0
```

- Flash the board and push the BOOT button to launch the programming

```
esp32:~/esp/esp-idf/examples/get-started/hello_world$ idf.py -p /dev/ttyUSB0
    flash
esptool.py -p /dev/ttyUSB0 -b 460800 --before default_reset --after
    hard_reset write_flash --flash_mode dio --flash_freq 40m --flash_size 2MB
     0x8000 partition_table/partition-table.bin 0x1000 bootloader/bootloader.
    bin 0x10000 hello-world.bin
esptool.py v2.8
Serial port /dev/ttyUSB0
Connecting........_____....._
Detecting chip type... ESP32
Chip is ESP32-PICO-D4 (revision 1)
...
Compressed 147232 bytes to 76527...
Wrote 147232 bytes (76527 compressed) at 0x00010000 in 1.7 seconds (effective
     675.4 kbit/s)...
Hash of data verified.
Leaving...
Hard resetting via RTS pin...
Done
```

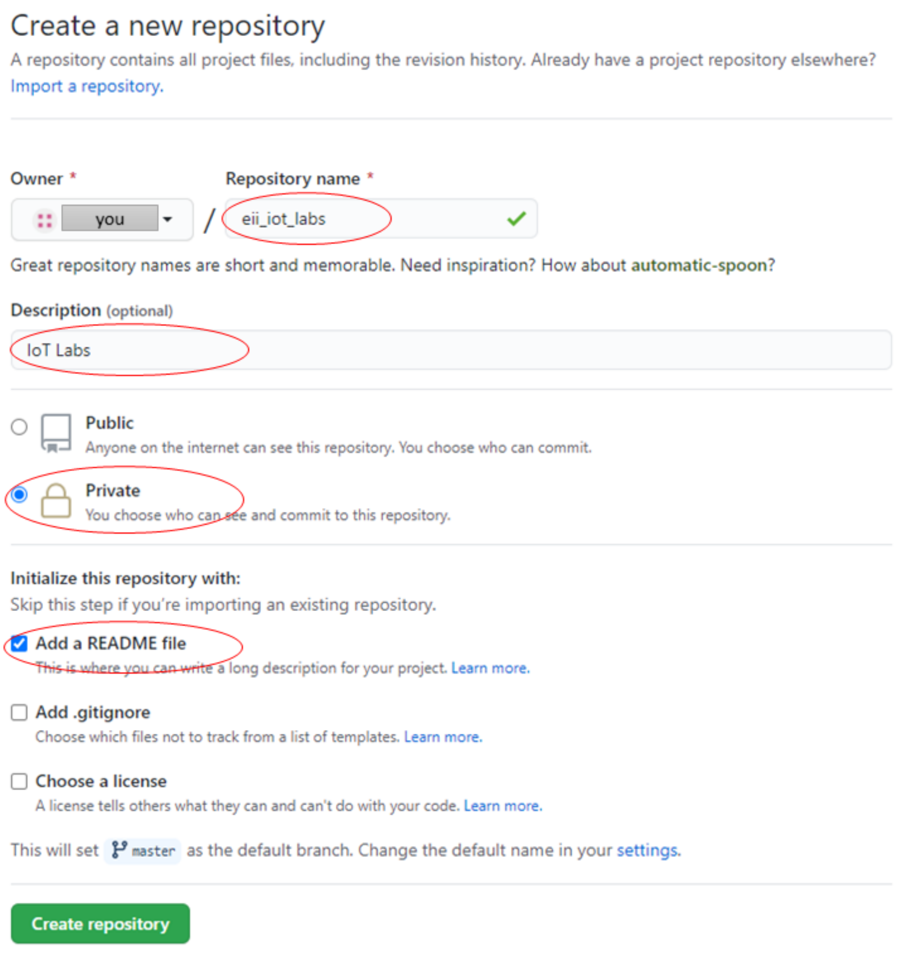
- Monitor console messages sent by the program running on the ESP32 card. To exit monitoring, typing « Ctrl+AltGr+] »

```
esp32:~/esp/esp-idf/examples/get-started/hello_world$ idf.py -p /dev/ttyUSB0
    monitor
...
I (294) spi_flash: flash io: dio
W (294) spi_flash: Detected size(4096k) larger than the size in the binary
    image header(2048k). Using the size in the binary image header.
I (304) cpu_start: Starting scheduler on PRO CPU.
I (0) cpu_start: Starting scheduler on APP CPU.
Hello world!
This is ESP32 chip with 2 CPU cores, WiFi/BT/BLE, silicon revision 1, 2MB
    embedded flash
Restarting in 10 seconds...
Restarting in 9 seconds...
Restarting in 8 seconds...
```

1.2 Creation of GitHub repository for Labs

Firstly, in the WEB interface of GitHub (open with Google Chrome or another navigator), you have to create a new repositories in GitHub, for example « labs ». **Configure your GitHub in private access** with a README.md file as shown the figure 1.2

Secondly, in a terminal, follow the steps to clone your new GitHub repositories in your computer :



Create a new repository

A repository contains all project files, including the revision history. Already have a project repository elsewhere? Import a repository.

Owner * / Repository name *

you / eii_iot_labs

Great repository names are short and memorable. Need inspiration? How about automatic-spoon?

Description (optional)

IoT Labs

Public
Anyone on the internet can see this repository. You choose who can commit.

[✓ Private]
You choose who can see and commit to this repository.

Initialize this repository with:
Skip this step if you're importing an existing repository.

[✓] Add a README file
This is where you can write a long description for your project. Learn more.

[ ] Add .gitignore
Choose which files not to track from a list of templates. Learn more.

[ ] Choose a license
A license tells others what they can and can't do with your code. Learn more.

This will set master as the default branch. Change the default name in your settings.

Create repository

FIGURE 1.2 – *Create a GitHub repository.*

1. You must configure your name and email address for GIT.
```
esp32:~/$ git config --global user.name "your name"
esp32:~/$ git config --global user.email "your email address"
```

2. You can now clone your « labs » repository to the computer. To obtain the URL of your repository, copy the repository URL located on GitHub webpage. The example below shows you the principle when you have to replace <your owner> by your GitHub owner.

```
esp32:~/$ git clone https://github.com/<your owner>/labs
```

3. You must enter this command in the new « labs » repository to avoid typing your username and password each time in Visual Studio Code.
```
esp32:~/$ cd labs
esp32:~/labs$ git config credential.helper store
```

You have information for configuring GIT for your new project.

1.3 Visual Studio Code with ESP-IDF

In order to develop applications in a user-friendly way, we use Microsoft Visual Studio Code throughout these Labs. Moreover, we use a Visual Studio Code project template located in GitHub : `https://github.com/fmuller-pns/esp32-vscode-project-template`.

To use the template :

- Go to your repository where we will create the first lab named « part1_iot_framework ».

```
esp32:~$ cd labs
esp32:~/labs$ mkdir part1_iot_framework
esp32:~/labs$ cd part1_iot_framework
```

- Clone the template project named « Visual Studio Code Template for ESP32 »

```
esp32:~/labs/part1_iot_framework$ git clone https://github.com/fmuller-pns/
   esp32-vscode-project-template.git
Cloning into 'esp32-vscode-project-template'...
remote: Enumerating objects: 30, done.
remote: Counting objects: 100% (30/30), done.
remote: Compressing objects: 100% (23/23), done.
remote: Total 30 (delta 8), reused 23 (delta 4), pack-reused 0
Unpacking objects: 100% (30/30), done.
```

- List working directory

```
esp32:~/labs/part1_iot_framework$ ll
total 12
drwxr-xr-x  3 esp32 esp32 4096 mai   26 16:17 ./
drwxr-xr-x 24 esp32 esp32 4096 mai   26 16:16 ../
drwxr-xr-x  5 esp32 esp32 4096 mai   26 16:18 esp32-vscode-project-template/
```

- Rename the folder.

```
esp32:~/labs/part1_iot_framework$ mv esp32-vscode-project-template
   lab1_framework
```

- Delete *.git* folder of the new project « lab1_framework ». Be careful, **do not delete the *.git* folder** located in the « labs » folder.

```
esp32:~/labs/part1_iot_framework$ cd lab1_framework
esp32:~/labs/part1_iot_framework/lab1_framework$ rm -fR .git
```

- Open Visual Studio Code for the new project.

```
esp32:~/labs/part1_iot_framework/lab1_framework$ code .
```

- Follow the section Getting Started to run the program on the board.
- Change the message in the *app_main()* function located in the *main.c* file.
- Build and run the program.
- You must commit and push the modification in GitHub. Follow the section Using GitHub with Visual Studio Code to do it.

Now, you are ready to use Visual Studio Code with ESP-IDF for other projects !

Part III

FreeRTOS

LAB N° 1

Task & Scheduling

Lab Objectives

- Creating, suspending and deleting a task.
- Understanding the scheduling of tasks and priority effect.
- Multi-cores scheduling
- Using Idle task.
- Task handler

1.1 Task scheduling on one core (Lab1-1)

The entry point of an application is the *app_main()* function. This function is actually a task of priority 1. We will usually create the other application tasks from the *app_main()* task. We will create an application with 2 tasks (including *app_main()* task), then 3 tasks running on 1 core in order to understand the behavior of the scheduler.

1.1.1 Understanding the task scheduling

1. Create the « lab1-1_1_core_sched » lab from « esp32-vscode-project-template » GitHub repository.
2. Overwrite the « main.c » file by the provided code of the « lab1-1_main.c » file.
3. Copy the provided « my_helper_fct.h » file to the « main » folder.
4. Copy the provided « lab1-1_sdkconfig.defaults » file to the project folder and rename « sdkconfig.defaults ».
5. Answer the following questions :

- Study the parameters of the following function : *xTaskCreate()* and *vTaskDelete()*. Web help

- Study the following function : *uxTaskPriorityGet()*. Web help

- Study the following function : *vTaskDelay()*. Web help

- What do they do the following macros ? *DISPLAY()* and *DISPLAYB()*.

- *Task 1* is an instance of the *vTaskFunction()*. Is that true ?

6. Open the « sdkconfig.defaults » file. What does the *CONFIG_FREERTOS_UNICORE* parameter correspond to ? Web help

7. What is the tick period in ms ? see the *CONFIG_FREERTOS_HZ* parameter and Web help

8. Build and run the program.
9. Trace in the figure 1.1 the behavior of the *app_main()* and *Task 1* tasks until 160 ticks.
10. Why the *app_main()* task does not run ?

11. Create a new instance *Task 2* of the *vTaskFunction()* with the same priority of *Task 1*, i.e. priority=5.

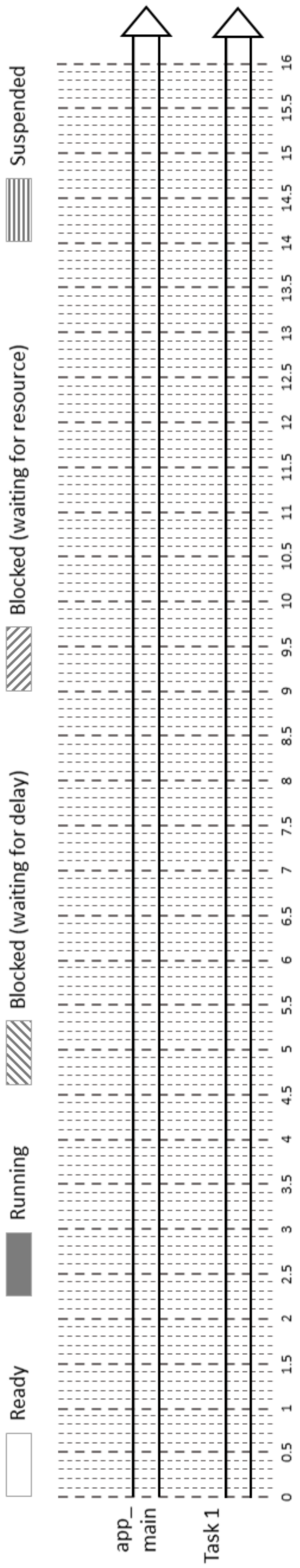


Figure 1.1 – *Scheduling of 2 tasks.*

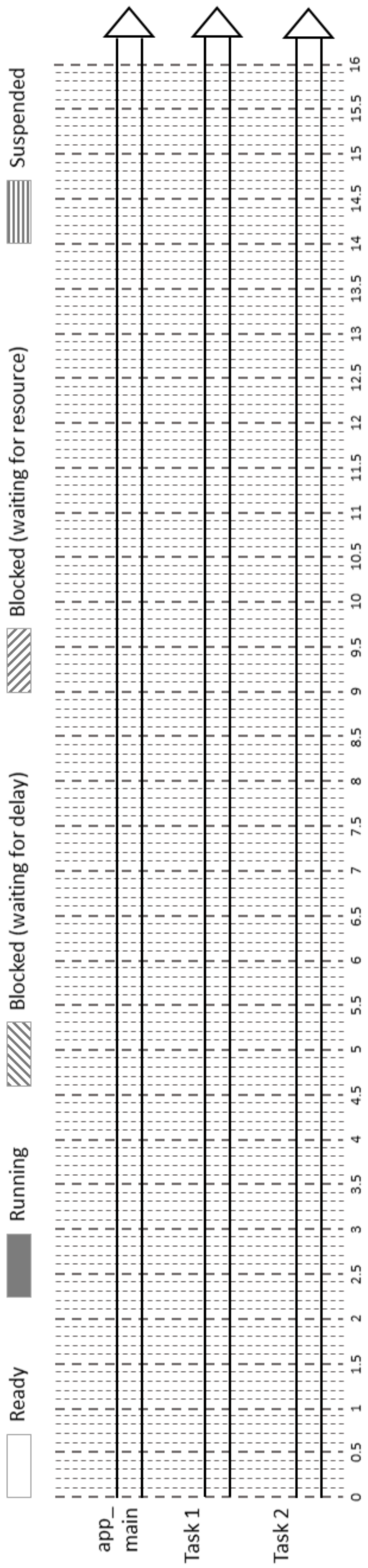


Figure 1.2 – *Scheduling of 3 tasks.*

12. Run the program. Comment on the behavior.

13. We are now going to suspend the scheduler with 2 functions when creating the tasks.

```
void app_main(void) {
  ...
  vTaskSuspendAll();
  // Task creation ...
  ...
  xTaskResumeAll();
  ...
```

14. Run the program.

15. For this scenario, trace in the figure 1.2 the behavior of the *app_main()*, *Task 1* and *Task 2* tasks until 160 ticks.

16. Comment on the behavior. Why the *app_main()* task does not run ?

17. Change the priority=6 for the Task 2. Run the program and comment on the behavior for this scenario.

18. We are now going to add delay on *vTaskFunction()* after the simulation of computation time.

```
void vTaskFunction(void *pvParameters) {
...
for (;; ) {
  DISPLAY("Run computation of %s", pcTaskName);

  /* Delay for simulating a computation */
  for (ul = 0; ul < mainDELAY_LOOP_COUNT; ul++){
  }

  // Add Delay of 300 ms
  DISPLAY("Delay of %s", pcTaskName);
  vTaskDelay(300 / portTICK_PERIOD_MS);
  ...
```

19. Trace in the figure 1.3 the behavior of the *app_main()*, *Task 1* and *Task 2* tasks until 160 ticks.

1.1.2 Using trace information

It is possible to get information about the tasks of the application using the *vTaskList()* function (Web help).

1. Copy the code provided in the « lab1-1_add_vtasklist.c » file to print trace information as below. Note that size of the buffer is approximately 40 bytes per task.

```
/* Buffer to extract trace information */
static char buffer[40*8];

void app_main(void) {
  ...
  /* Print task list */
  vTaskList(buffer);
  printf("----------------------------------------------------------\n"
     );
  printf("task\t\tstate\tprio\tstack\ttsk id\tcore id\n");
  printf("----------------------------------------------------------\n"
     );
  printf(buffer);

  DISPLAY("==== Exit APP_MAIN ====");
  ...
```

2. The 2 first parameters are compulsory to use the *vTaskList()* function. The last parameter is used to print the code id. We active the 3 parameters. 2 solutions :
   - Using the menuconfig command and go to *Component config/FreeRTOS/Enable FreeRTOS trace facility, Enable FreeRTOS stats formatting functions ans Enable display of xCoreID in vTaskList.*
   - Add these lines below in the « sdkconfig.defaults » file (provided in *lab1-1_config_trace_information.txt* file)and delete the « sdkconfig » to take into account the modifications.

   ```
   # Trace facility to extract trace information
   CONFIG_FREERTOS_USE_TRACE_FACILITY=y
   CONFIG_FREERTOS_USE_STATS_FORMATTING_FUNCTIONS=y
   CONFIG_FREERTOS_VTASKLIST_INCLUDE_COREID=y
   ```

3. Run the program. Copy the trace information. The *stack* column is the amount of free stack space in bytes there has been since the task was created.

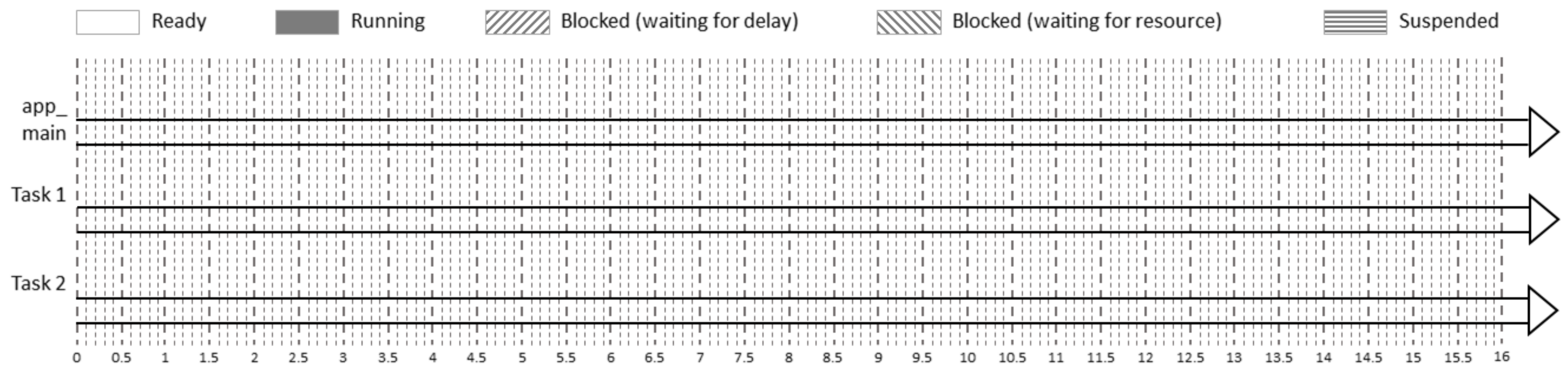


Figure 1.3 – *Scheduling of 3 tasks with task delay.*

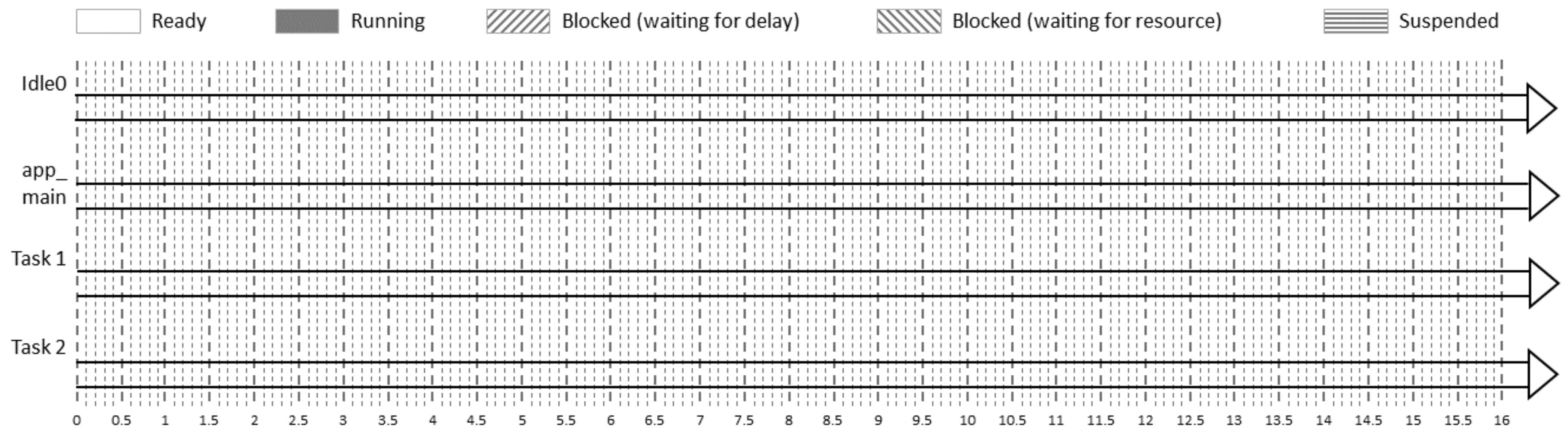


Figure 1.4 – *Scheduling of 3 tasks with task delay.*

4. How many tasks are there ?

5. What is the highest priority task ?

6. What does the *main* task correspond to ?

7. What is the role of the *ipc0* task ? Web help

8. What is the role of the *esp_timer* task ? Web help

9. What is the role of the *Tmr Svc* task ? Web help 1 and Web help 2

10. What is the role of the *IDLE0* task ? What is its priority ? Web help

11. The default stack size is 4000 bytes. Change the constant *STACK_SIZE*=1400 bytes. What is the problem ? Change the size to the nearest hundredth to avoid the problem.

1.1.3 Idle and Tick Task Hooks

An idle task hook is a function that is called during each cycle of the idle task. To use them, we will modify the configuration.

1. Add the two hook functions provided in the « lab1-1_add_idle_hook.c » file.

```
bool vApplicationIdleHook ( void ){
  DISPLAY ("IDLE");
  return true;
}
```

```
void vApplicationTickHook ( void ){
}
```

2. Why are there 2 hook functions ? What is the difference ? Web help 1 and Web help 2

3. The first parameter is compulsory to use the hooks functions. The second parameter adjusts the stack size depending on the behavior of the hook functions. 2 solutions :
   - Using the menuconfig command and go to *Component config/FreeRTOS/Use FreeRTOS legacy hooks.*
   - Add these lines below in the « sdkconfig.defaults » file (provided in *lab1-1_config_idle_hook.txt* file) and delete the « sdkconfig » to take into account the modifications.

   ```
   # allows add hook functions
   CONFIG_FREERTOS_LEGACY_HOOKS=y
   CONFIG_FREERTOS_IDLE_TASK_STACKSIZE=4096
   ```

4. Run the program. What do you see for the *Idle* and *app_main* tasks ?

5. Modify the task delay of vTaskFunction() function to 400ms instead of 100ms. What do you see for *Idle* and *app_main* tasks ? Conclusion ?

6. Trace in the figure 1.4 the behavior of tasks until 160 ticks.

1.2 Task scheduling on two cores (Lab1-2)

the ESP32 has 1 or 2 cores. It is possible to choose on which core the task should be mapped or to let the scheduler choose. At the end of this Lab, we will see another solution to use the *idle* task without using the configuration but an API.

1. Create the « lab1-2_2_cores_sched » lab from « esp32-vscode-project-template » GitHub repository.
2. Overwrite the « main.c » file by the provided code of the « lab1-2_main.c » file.
3. Copy the provided « my_helper_fct.h » file to the « main » folder.
4. Copy the provided « lab1-2_sdkconfig.defaults » file to the project folder and rename « sdkconfig.defaults ».

5. Study the following function : *xTaskCreatePinnedToCore()*. Web help

1.2.1 Task scheduling scenarios

We are going to test different scheduling scenarios according to the cores on which the tasks will be executed. In order to facilitate the different scenarios to be executed, we will use the preprocessor macros as below :

```
//#define DIFF_PRIORITY

#define PINNED_TO_CORE 0x00
//   0x00: Task 1: Core 0, Task 2: Core 0
//   0x01: Task 1: Core 0, Task 2: Core 1
//   0x10: Task 1: Core 1, Task 2: Core 0
//   0x11: Task 1: Core 1, Task 2: Core 1

//#define IDLE_HOOKS

//#define TASK_DELAY
```

You must run the 4 scenarios, copy the trace till 160 ticks (1600 ms) and comment it.

- Uncomment *DIFF_PRIORITY* to have the priority(Task 1) = 5 and priority(Task 2) = 6.
- Scenario n° 1 : Task 1 : Core 0, Task 2 : Core 0

- Scenario n° 2 : Task 1 : Core 0, Task 2 : Core 1

- Scenario n^o 3 : Task 1 : Core 1, Task 2 : Core 0

- Scenario n^o 4 : Task 1 : Core 1, Task 2 : Core 1.

For the last scenario n^o 5, we are going to add a task on *core 0* using task mapping of scenario n^o 4.

1. Add a *Task 3* on core 0 with priority = 1.
2. Trace in the figure 1.5 the behavior of the tasks until 160 ticks.

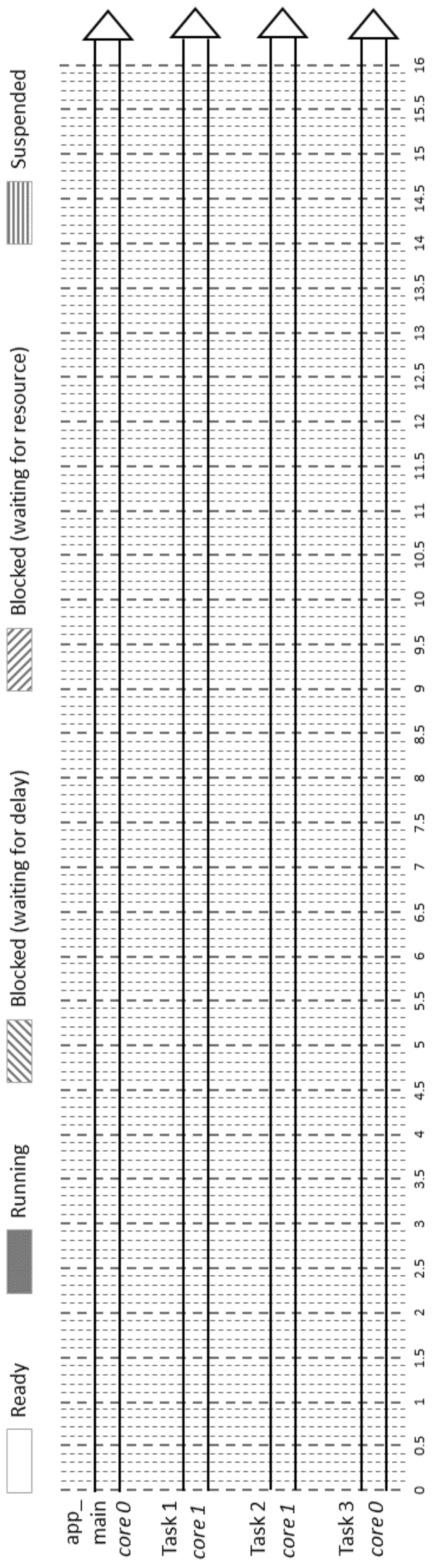


Figure 1.5 – *Scheduling on 2 cores with 4 tasks.*

1.2.2 Idle Task Hook from API (Optional work)

The disadvantage of configuring the hook functions from the configuration menu is the declaration of the 2 functions while generally the *vApplicationIdleHook()* function is the only useful function. Thus, it is possible to use an API which allows adding only the *vApplicationIdleHook()* function.

1. Uncomment the *IDLE_HOOKS* preprocessor macro. We use the previous scenario n° 5.
2. Study the following function : *esp_register_freertos_idle_hook_for_cpu()*. Web help
3. Run the program and comment the behavior. What do you notice about the execution of the *IDLE1* task ?
4. Uncomment the *TASK_DELAY* preprocessor macro.
5. Run the program and comment the new behavior, in particular the *IDLE1* task.

1.3 Approximated/Exactly periodic task (Lab1-3)

In some applications it is important to have tasks with an exact period. We will compare the use of the *TaskDelay()* function which allows you to wait for a certain time and the *TaskDelayUntil()* function which allows you to adjust the delay according to the time spent.

1. Duplicate the « lab1-2_2_cores_sched » folder to « lab1-3_periodic_task ».
2. Check that you have uncommented the *DIFF_PRIORITY*, *IDLE_HOOKS* and *TASK_DELAY* preprocessor macros.
3. Set the *PINNED_TO_CORE* macro to $0x00$ to run all tasks on *PRO_CPU* (Core 0).
4. Change the behavior of the *vTaskFunction()* task as below.

```
#ifdef TASK_DELAY
  /* Approximated/Periodic Delay */
  #ifdef PERIODIC_TASK_DELAY
    DISPLAY("Periodic Delay of %s", pcTaskName);
    vTaskDelayUntil(&xLastWakeTime, pdMS_TO_TICKS(300));
  #else
    DISPLAY("Approximated Delay of %s", pcTaskName);
    vTaskDelay(pdMS_TO_TICKS(300));
  #endif
#endif
```

5. What does it do the following macro *pdMS_ TO_ TICKS()* ?

6. Study the *vTaskDelayUntil()* function. Web help

7. Run the program and comment the behavior. What is the interval/period of Tasks 1,2 and 3 ? To compute time, use the tick number from the display of « Run computation of Task X » (X = 1,2 or 3) comment.

8. Uncomment the *PERIODIC_TASK_DELAY* preprocessor macro.
9. Build the project and correct the compilation error if necessary.
10. Run the program at least up to 400 ticks and comment the behavior. What is the interval/period of Tasks 1,2 and 3 ? What is the problem ?

11. Change the *mainDELAY_ LOOP_ COUNT* constant to *0x1FFFF*. What is the interval/period of Tasks 1,2 and 3 ? Comment the new behavior.

1.4 Task handler and dynamic priority (Lab1-4, Optional work)

During the execution of the application, it is sometimes useful to modify the priority of a task or any other actions. To do this, it is necessary to get the handler of the task on which we have to operate these actions.

1. Create the « lab1-4_task_handler_priority » lab from « esp32-vscode-project-template » GitHub repository.
2. Overwrite the « main.c » file by the provided code of the « lab1-4_main.c » file.

3. Copy the provided « my_helper_fct.h » file to the « main » folder.
4. Copy the provided « lab1-4_sdkconfig.defaults » file to the project folder and rename « sdkconfig.defaults ».
5. Study the *uxTaskPriorityGet()* and *vTaskPrioritySet()* functions. Web help
6. Study the *vTaskGetRunTimeStats()* function. What parameters should be configured in *sdkconfig* file or from graphical menu config ? Web help
7. Explain the use of the handler. In what tasks ? Why ?
8. Run the program. What is the problem ?
9. How are you going to correct the problem ?
10. Run the program after code modification and comment the scenarios.
11. Comment also the statistics.

LAB N° 2

Message Queue & Interrupt service

Lab Objectives

- Using message queue API.
- Using message queue with interrupts

2.1 Single Message Queue (Lab2-1)

We want to create 3 tasks. The functions implementing the tasks will be named *Task1()* (priority=2, periodic : 500ms, running on the *Core 1*), *Task2()* (priority=2, , running on the *Core 0*) and *Task3()* (priority=3, , running on the *Core 0*) respectively. *Task1()* will send a number (using a message queue object) to the *Task2()*. Each task displays string to the terminal with *DISPLAY()* or *DISPLAYI()* macros.

1. Create the « lab2-1_single_msg_queue » lab from « esp32-vscode-project-template » GitHub repository.
2. Copy the provided « my_helper_fct.h » file to the « main » folder.
3. Copy the provided « lab2-1_sdkconfig.defaults » file to the project folder and rename « sdkconfig.defaults ».
4. Append the content of *add_includes.c* file to the start of the *main.c* file.
5. Study the following function : *xQueueSend()*, *xQueueReceive()*. Web help 1, Web help 2
6. Write the 3 tasks with empty body.

7. Declare the 3 tasks on the *app_main()* function (Best practice : use constants for priorities, stack size, ...).
8. Add the message queue (depth is 5) in the program. The type of the message is *uint32_t*.
9. Write the behavior of the tasks as below :
   - Task 3 blocks for 100ms, displays string to the terminal and then simulates a computation of 20ms (2 ticks).

     ```
     // Task blocked during 100 ms
     DISPLAY(...);
     vTaskDelay(pdMS_TO_TICKS(100));
     DISPLAY(...);
     // Compute time : 2 ticks or 20 ms
     COMPUTE_IN_TICK(2);
     ```

   - The *Task 1* should be periodic with a periodicity of 500ms. It posts in a message queue to the *Task 2*, check the result of the post (Failed or posted message), simulates a computation of 40ms (4 ticks) and wait for next period. Note that the write function in the queue should not be blocking.

     ```
     // Post
     uint32_t result = xQueueSend(...);
     // Check result
     if (result) ...
     // Compute time : 4 ticks or 40 ms
     COMPUTE_IN_TICK(4);
     // block periodically : 500ms
     vTaskDelayUntil(...);
     ```

   - Task 2 waits for a message through the message queue, displays the task number and message received and then simulates a computation of 30ms (3 ticks).

     ```
     // Wait for message
     ...
     // display task number and message
     DISPLAY(...);
     // Compute time : 3 ticks or 30 ms
     COMPUTE_IN_TICK(3);
     ```

   - Don't forget to create a message queue.
10. Build and run the program.
11. Trace in the figure 2.1 the behavior of the 3 tasks until 160 ticks.
12. What is the period of the task 2 ?

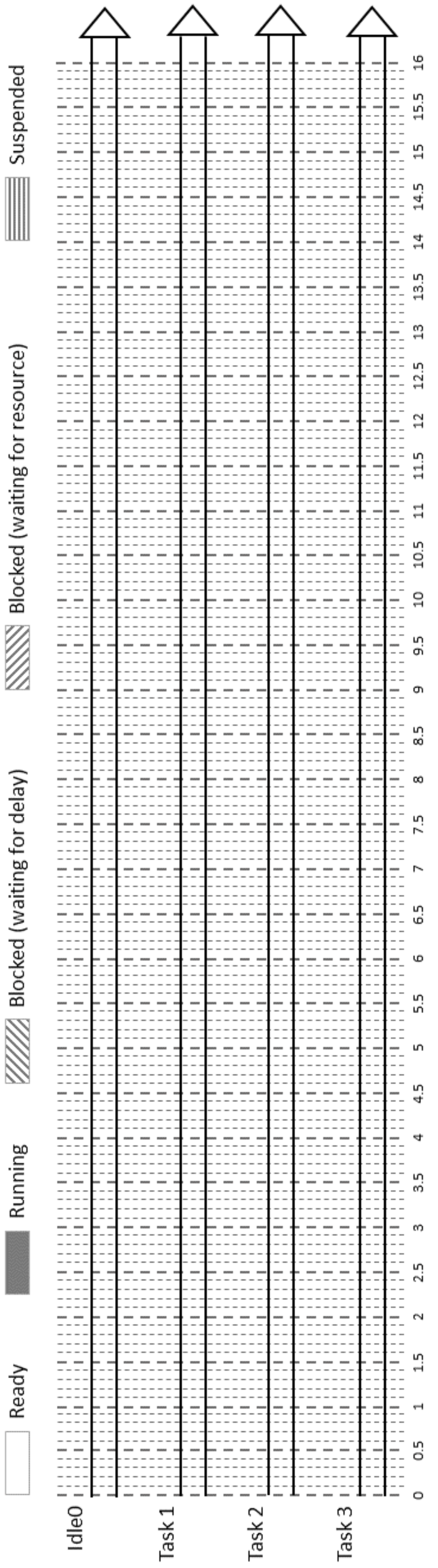


Figure 2.1 – *Message queue and 3 tasks.*

2.2 Message Queue with time out (Lab2-2)

The concept of timeout only applies to blocking system calls. When a task is blocked, it will wake up (go to the ready state) automatically after a period of time called Timeout, even if the expected event has not happened.

- Duplicate the « lab2-1_single_msg_queue » folder to « lab2-2_single_msg_queue_timeout ».
- Force *Task 2* to wake up every 300ms using the Timeout associated with the *xQueueReceive()* function as below.

```
static const char* TAG = "MsgTimeOut";
...
if (xQueueReceive(...)) {
  DISPLAYI(TAG, "Task 2, mess = %d", value);
  COMPUTE_IN_TICK(3);
}
else {
  DISPLAYE(TAG, "Task 2, Timeout!");
  COMPUTE_IN_TICK(1);
}
```

- Build and run the program.
- Trace in the figure 2.3 the behavior of the 3 tasks until 160 ticks.

2.3 Blocking on single Queue (Lab2-3, Optional work)

We want to illustrate the problems of writing/reading queue messages and the impact on the scheduling of tasks. The figure 2.2 illustrates the application. All tasks run on the *Core 0*. The message queue contains items of integer type.

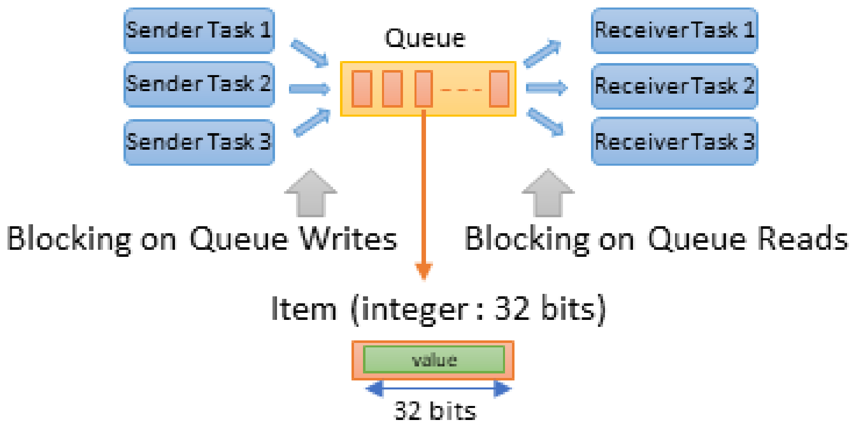


FIGURE 2.2 – *Blocking on single queue.*

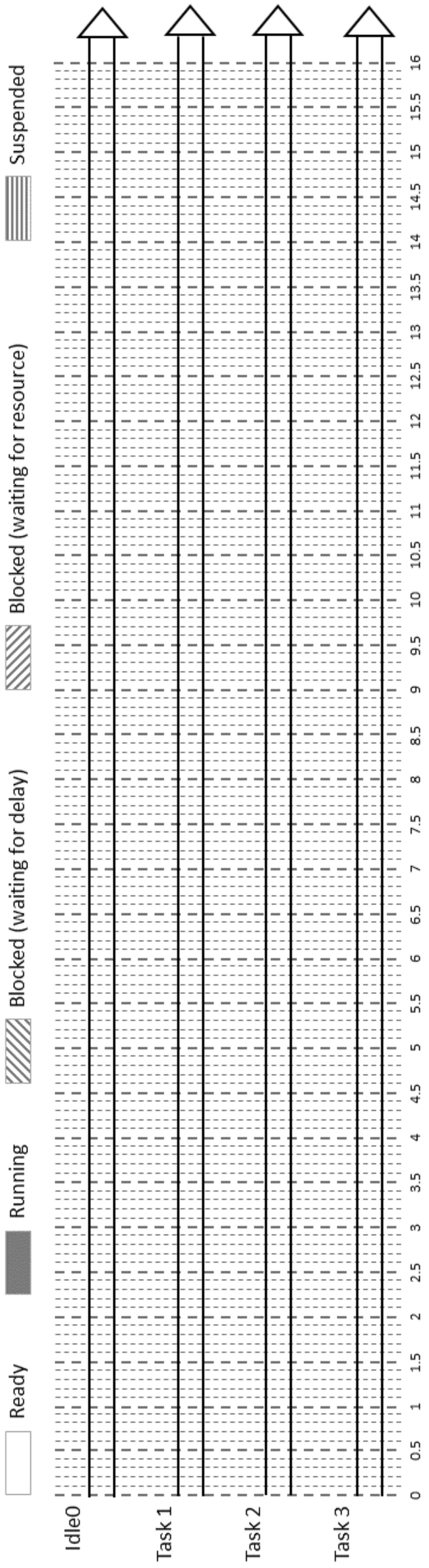


Figure 2.3 – *Message queue with timeout and 3 tasks.*

2.3.1 Blocking on Queue Reads

1. Create the « lab2-3_single_msg_queue_blocked » lab from « esp32-vscode-project-template » GitHub repository.
2. Copy the provided « my_helper_fct.h » file to the « main » folder.
3. Copy the provided « lab2-3_sdkconfig.defaults » file to the project folder and rename « sdkconfig.defaults ».
4. Overwrite the « main.c » file by the provided code of the « lab2-1_main.c » file.
5. Complete the code. For the moment, initialize the constants as below :

```
const uint32_t SENDER_TASK_PRIORITY = 3;
const uint32_t RECEIVER_TASK_PRIORITY = 3;
const uint32_t MESS_QUEUE_MAX_LENGTH = 10;
const uint32_t SENDER_TASK_NUMBER = 3;
const uint32_t RECEIVER_TASK_NUMBER = 3;
```

6. Build and run the program.
7. Explain the behavior.
8. Modify the priority of *receiver task* to 5 and run the program.
9. Explain the new behavior.
10. Modify the priority of *sender task* to 5 and the priority of *receiver task* to 3. Run the program.
11. Explain the new behavior.

2.3.2 Blocking on Queue Writes

1. Modify the priority of *sender task* to 5 and the priority of *receiver task* to 3.
2. Modify the capacity of the message queue to 2 (i.e. $MESS\_QUEUE\_MAX\_LENGTH = 2$)

3. Explain the problem.

4. How to correct the problem by adjusting the priority of tasks ?

2.4 Using message queue with interrupts (Lab2-4)

The figure 2.4 illustrates the application we want to achieve. When we press the push button, it will trigger on falling edge an interrupt (*Push_ button_ isr_ handler()*) that will send a message containing the GPIO pin number of push button to the *vCounterTask()* task. If the button is not pressed after 5 seconds, a message is displayed indicating that the button must be pressed to trigger an interrupt. A simple way is to use the timeout of the message queue received function. All tasks run on the *Core 0*.

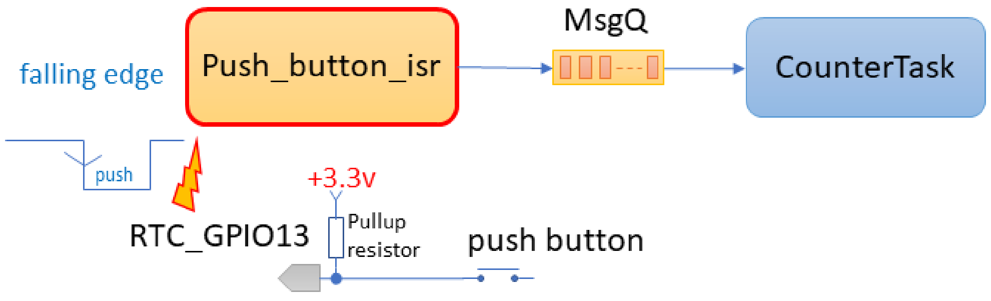


FIGURE 2.4 – *Interrupt application with message queue.*

1. Create the « lab2-4_single_msg_queue_interrupt » lab from « esp32-vscode-project-template » GitHub repository.
2. Overwrite the « main.c » file by the provided code of the « lab2-4_main.c » file.
3. Copy the provided « my_helper_fct.h » file to the « main » folder.
4. Copy the provided « lab2-4_sdkconfig.defaults » file to the project folder and rename « sdkconfig.defaults ».
5. Study the following function : *xQueueSendFromISR()*. Web help

6. Study the following function : *uxQueueMessagesWaiting()*. Web help

7. Configure the GPIO for the push button (RTC_GPIO13) as below. Note that the RTC_GPIO13 is the IO15 pin in the board.

```
/* Config GPIO */
gpio_config_t config_in;
config_in.intr_type = GPIO_INTR_NEGEDGE;  // falling edge interrupt
config_in.mode = GPIO_MODE_INPUT;      // Input mode (push button)
config_in.pull_down_en = false;
config_in.pull_up_en = true;          // Pull-up resistor
config_in.pin_bit_mask = (1ULL<<PIN_PUSH_BUTTON); // Pin number
gpio_config(&config_in);
```

8. Write the creation of the message queue (item size = 5) and the *vCounterTask()* task in the *app_main()* function.
9. Complete the body of the *Push_button_isr_handler()* by following the comments. Declare an *isrCount* global variable to count each trigger of interrupt. The argument of the *Push_button_isr_handler()* interrupt is the GPIO pin number of push button.
10. Write the installation of interrupt service in the *app_main()* function as below :

```
/* Install ISR */
gpio_install_isr_service(0);
gpio_isr_handler_add(PIN_PUSH_BUTTON, Push_button_isr_handler, (void
    *)PIN_PUSH_BUTTON);
```

11. Write the body of *vCounterTask()* task by following the comments in the source code.
12. Perform the wiring on the board. The RTC_GPIO13 is the IO15 pin. Refer to the documentation *ESP32-PICO-D4_Pin_Layout.pdf.*
13. Build and run the program.
14. When is the message "number of items" displayed ? why ?

15. Explain how to correct the problem ? more than one solution is possible.

16. Change the program according to your solution. Build and run the program.

LAB N° 3

Semaphore & Mutex

Lab Objectives

- Using semaphore API.
- Using Mutex

3.1 Specification of the application

We want to implement the functional structure (cf. figure 3.1) using FreeRTOS.

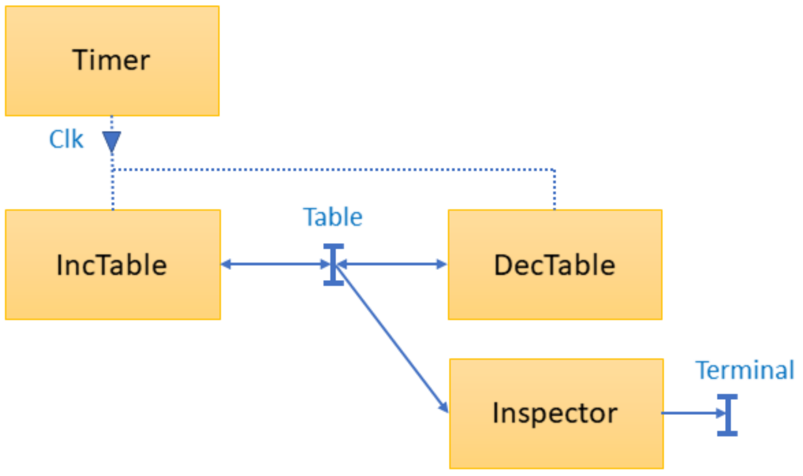


FIGURE 3.1 – *Functional description.*

The behavioral description of tasks is presented below in algorithm form. *Table* is an array of integer values that can represent a signal, such as a ramp. Its size is a constant (TABLE_SIZE) which can be modified during the tests. During initialization, it takes the

values [0,1,2,. . ., TABLE_SIZE-1]. Use TABLE_SIZE = 400.
Do not forget to use the *pdMS_TO_TICKS(iTime time)* macro to convert time in millisecond to tick number.

3.1.1 Timer task

This is a task that performs a *TaskDelay()* and generates the synchronization (semaphore) via *Clk*. The function *computeTime()* simulates a execution time in millisecond.

```
Task Timer is
    Properties: Priority = 5
    Out        : Clk is event
    Cycle :
        waitForPeriod(250 ms);
        computeTime(20 ms);
        print("Task Timer : give sem");
        notify(Clk);
    end
end
```

Algorithm 1: Timer algorithm.

3.1.2 IncTable task

It is a temporary action activated by *Clk*. An *constNumber* is passed by the parameters of the task. Every 5 activations, the task increments by the *constNumber* for each element of *Table*. We simulate a computation time by the *computeTime()* pseudo function. Its functional behavior is as follows :

```
Task IncTable is
    Properties: Priority = 4
    In          : Clk is event
    In/Out      : Table is array[0 to TABLE_SIZE-1] of integer
    ActivationNumber := 0;
    Cycle Clk :
        if ActivationNumber = 0 then
            for index :=0 to TABLE_SIZE-1 do
                Table[index] := Table[index] + constNumber;
            end
            computeTime(50 ms);
            ActivationNumber := 4;
        else
            ActivationNumber := ActivationNumber - 1;
        end
    end
end
```

Algorithm 2: IncTable algorithm.

3.1.3 DecTable task

Its functional behavior is as follows :

```
Task DecTable is
    Properties: Priority = 4
    In          : Clk is event
    In/Out      : Table is array[0 to TABLE_SIZE-1] of integer
    Cycle Clk :
        for index :=0 to TABLE_SIZE-1 do
            Table[index] := Table[index] - 1;
        end
        computeTime(50 ms);
    end
end
```

Algorithm 3: DecTable algorithm.

3.1.4 Inspector task

It is a task which constantly checks the consistency of the *Table* and displays an error message when an inconsistency is found in the *Table* (exit on the program, use *exit(1)* function). For this, it takes the first value of the *Table* as a reference (*reference* = *Table[0]*) and checks each element of *Table* in accordance with its reference (*Table[index]* = *reference* + *index*). When the *Table* has been fully browsed, the cycle begins again (a new reference is taken and the *Table* is checked again). Its functional behavior is as follows :

```
Task Inspector is
    Properties: Priority = 4
    In          : Clk is event
    In          : Table is array[0 to TABLE_SIZE-1] of integer
    ActivationNumber := 0;
    Cycle Clk :
        print("Task Inspector is checking.");
        reference := Table[0];
        error := false;
        for index :=1 to TABLE_SIZE-1 do
            smallComputeTime(100 us);
            if $Table[index] \neq (reference + index)$ then
                error := true;
            end
        end
        print("Task Inspector ended its checking.");
        if error = true then
            print(TAG, "Consistency error in the Table variable.");
            exit();
        end
    end
end
```

Algorithm 4: Inspector algorithm.

3.2 First Task synchronization (Lab3-1)

Firstly, we will only implement the *Timer*, *DecTable* and *IncTable* tasks as well as the *Clk* and *Table* relationships. The *computeTime()* function can be implemented by the *COMPUTE_IN_TICK()* macro (1 tick = 10ms) and the print() function by *DISPLAY()* macro.

3.2.1 Writing the application

1. Create the « lab3-1_one_sem_clk » lab from « esp32-vscode-project-template » GitHub repository.
2. Copy the provided « lab3-1_sdkconfig.defaults » file to the project folder and rename « sdkconfig.defaults ».
3. Copy the provided « my_helper_fct.h » file to the « main » folder.
4. Overwrite the « main.c » file by the provided code of the « lab3-1_main.c » file.
5. Write the program with the behavior of these 3 tasks and 1 semaphore (*xSemClk*) using the algorithms proposed above. All the tasks are created on the *Core_0*. Below is a creation reminder for a semaphore.

```
/* Creating Binary semaphore */
SemaphoreHandle_t xSemClk;
xSemClk = xSemaphoreCreateBinary();
/* Using semaphore */
xSemaphoreGive(xSemClk);
xSemaphoreTake(xSemClk, portMAX_DELAY);
```

6. Build the program without running it.

3.2.2 Scenarios with one clock semaphore

We will perform scenarios described in the Table 3.1 in order to identify problems and improve the program later. The task priority of *Timer* is 5 and run on *Core_ 0*.

Scenario	IncTable task	DecTable task
1	Prio(4),Core(0)	Prio(4),Core(0)
2	Prio(3),Core(0)	Prio(4),Core(0)
3	Prio(4),Core(0)	Prio(4),Core(1)
4	Prio(3),Core(0)	Prio(4),Core(1)

Table 3.1 – *Scenarios for task synchronization.*

Scenario 1 : Run the program, copy the console, trace in the figure 3.2 the behavior of the 3 tasks until 160 ticks and explain the problem.

Scenario 2 : Run the program, copy the console and explain the problem.

Scenario 3 : Run the program, copy the console and explain the problem.

Scenario 4 : Run the program, copy the console and explain the problem.

3.3 Task synchronization with 2 semaphores (Lab3-2)

We identified different issues in the previous section. We will perform again the scenarios described in the Table 3.1.

1. Duplicate the « lab3-1_one_sem_clk » folder to « lab3-2_two_sem_clk ».
2. Correct the clock program to send 2 separate semaphores (*xSemIncTab* and *xSemDecTab*) to *IncTable* and *DecTable* tasks.

Scenario 1 : Run the program, copy the console, trace in the figure 3.3 the behavior of the 3 tasks until 160 ticks and explain the behavior.

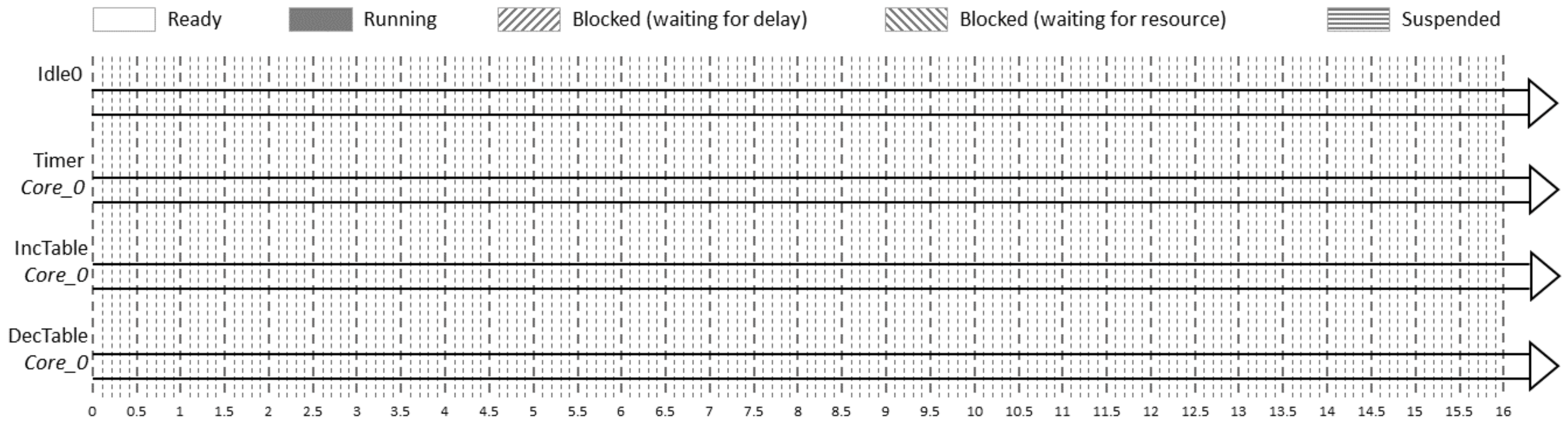


FIGURE 3.2 – *Scenario 1 : Priority(DecTable=4, IncTable=4), Core(DecTable=0, IncTable=0).*

Scenario 2 : Run the program, copy the console, trace in the figure 3.4 the behavior of the 3 tasks until 160 ticks and explain the behavior.

Scenario 3 : Run the program, copy the console, trace in the figure 3.5 the behavior of the 3 tasks until 160 ticks and explain the behavior.

Scenario 4 : Run the program, copy the console and explain the behavior.

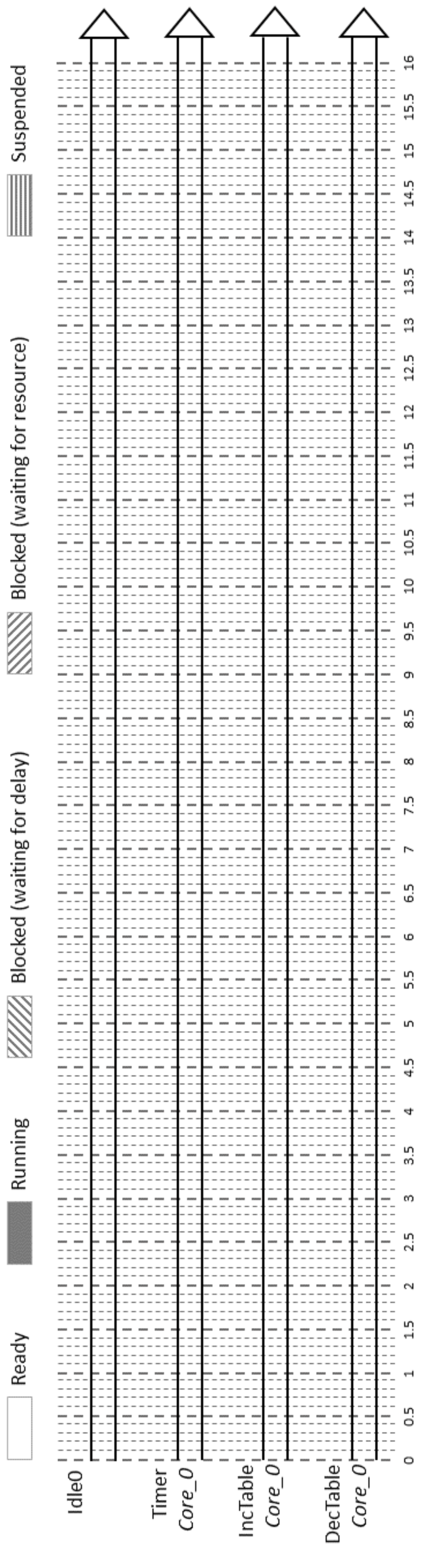


Figure 3.3 – *Scenario 1 : Priority(DecTable=4, IncTable=4), Core(DecTable=0, IncTable=0).*

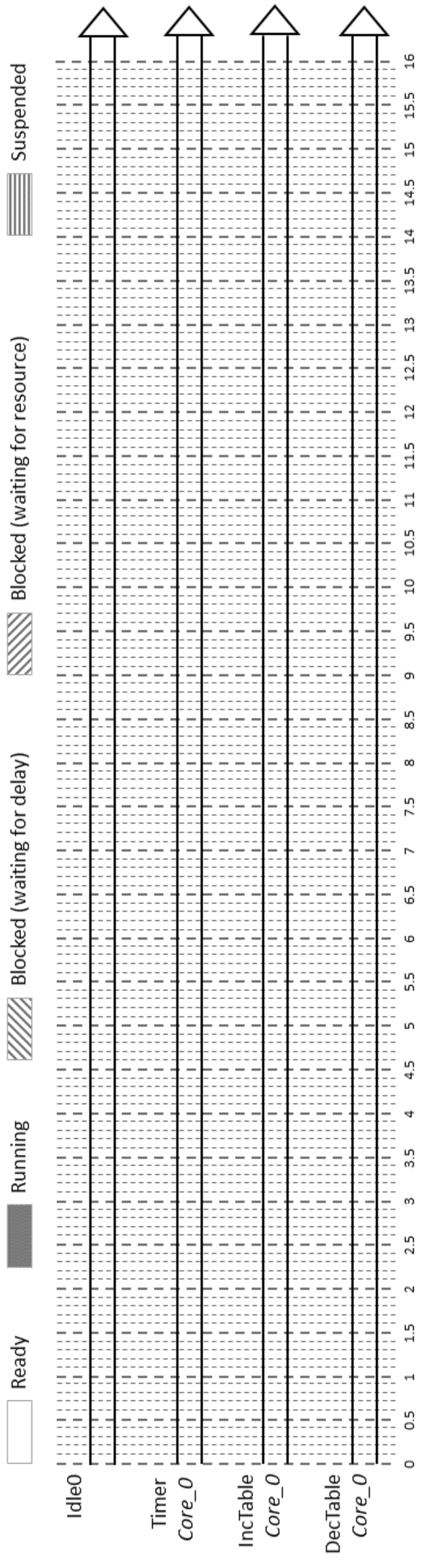


Figure 3.4 – *Scenario 2 : Priority(DecTable=4, IncTable=3), Core(DecTable=0, IncTable=0).*

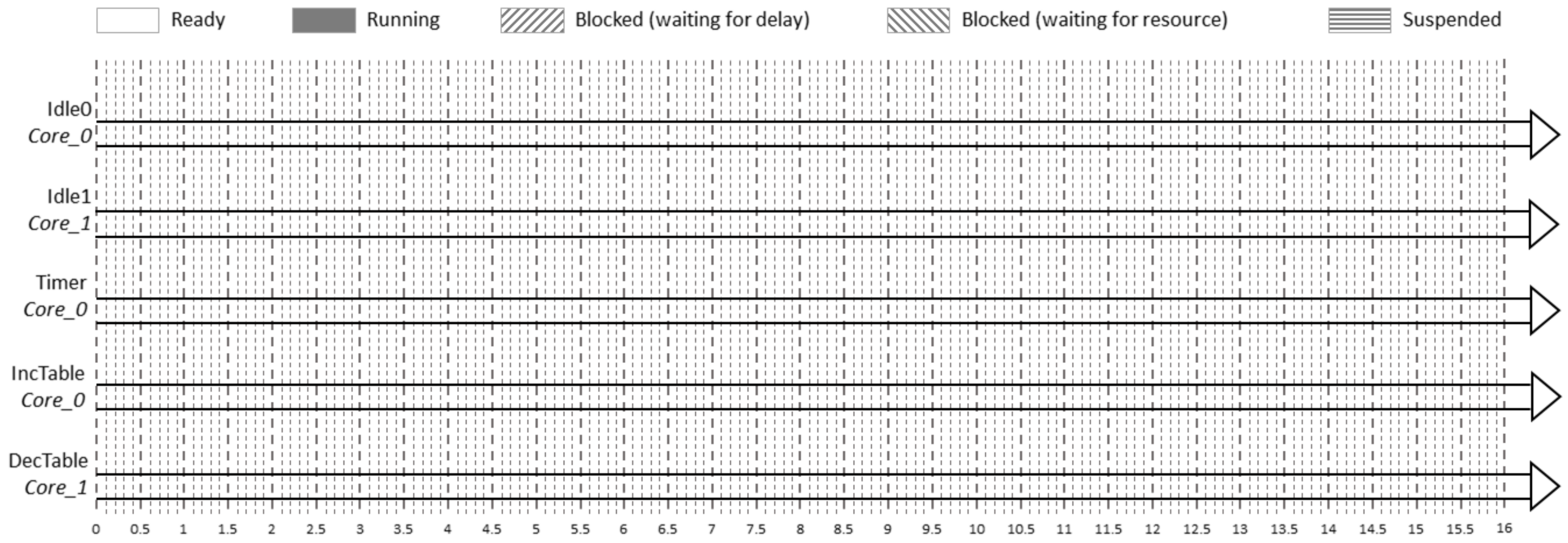


Figure 3.5 – *Scenario 3 : Priority(DecTable=4, IncTable=4), Core(DecTable=1, IncTable=0).*

3.4 Mutual Exclusion (Lab3-3)

We now add the *Inspector* task. We will perform the program with the task priorities described in the Table 3.2.

Timer task	IncTable task	DecTable task	Inspector task
Prio(5),Core(0)	Prio(3),Core(0)	Prio(3),Core(0)	Prio(4),Core(0)

Table 3.2 – *Task priorities with mutex.*

1. Duplicate the « lab3-1_one_sem_clk » folder to « lab3-2_two_sem_clk ».
2. Add the *Inspector* task.
3. Run the program, copy the console and explain the behavior. What is the problem ?

4. Correct the problem of initialization.
5. Run the program, copy the console and explain the behavior. What is the problem ?

6. Choose a new priority of the *Inspector* task to solve the problem.
7. Run the program, copy the console until 40 ticks, trace in the figure 3.6 the behavior of the 3 tasks until 40 ticks and explain the behavior.

8. Modify the *Inspector* task by adding a *Mutex* to manage access to the critical area. Below is a creation reminder for a Mutex.

```
/* Mutex */
SemaphoreHandle_t xSemMutex;
xSemMutex = xSemaphoreCreateMutex();
/* Using Mutex */
xSemaphoreGive(xSemMutex);
xSemaphoreTake(xSemMutex, portMAX_DELAY);
```

9. Run the program, copy the console, trace in the figure 3.7 the behavior of the 3 tasks until 40 ticks and explain the behavior and the effect of the Mutex.

10. Change the priority of *Inspector* task to 4. Run the program, copy the console until 40 ticks. What is the problem ?

11. We decide to change the *Inspector* task to *Core_ 1*. Run the program, copy the console until 40 ticks. What is the problem ?

12. We now decide to add a delay of 2 ticks after giving the mutex (using *vTaskDelay()* function). Run the program, copy the console, trace in the figure 3.8 the behavior of the 3 tasks and Mutex until 90 ticks and explain the behavior.

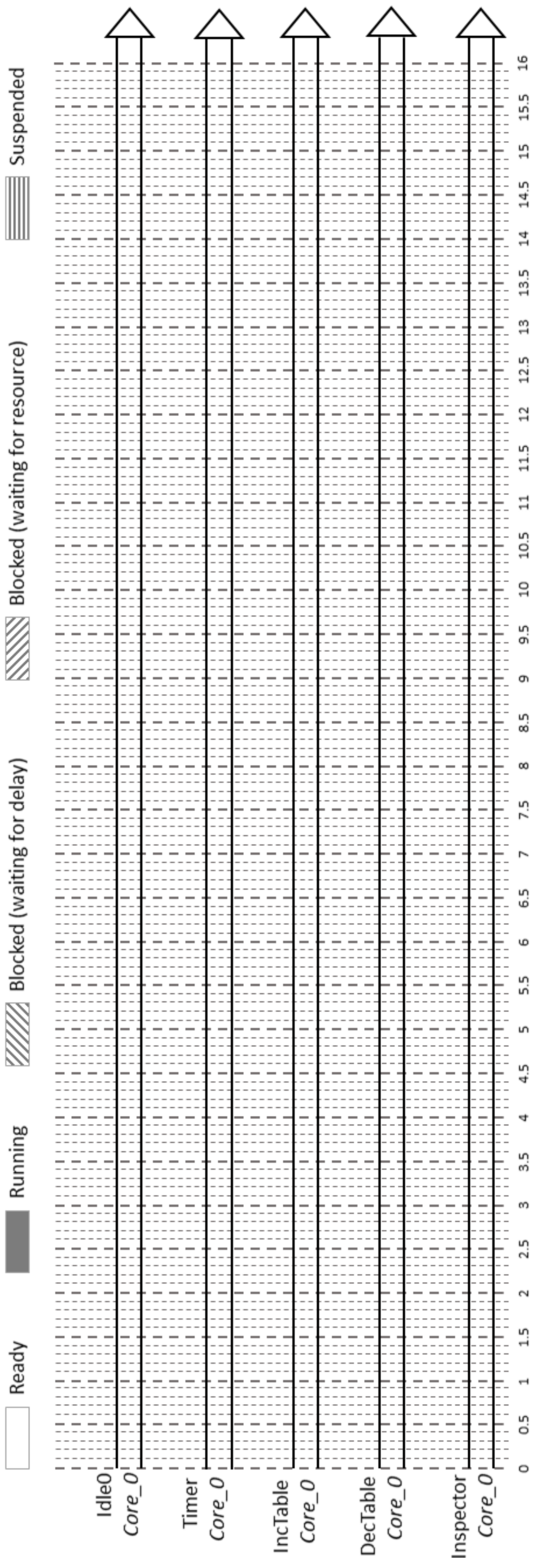


Figure 3.6 – *Program with Task inspector.*

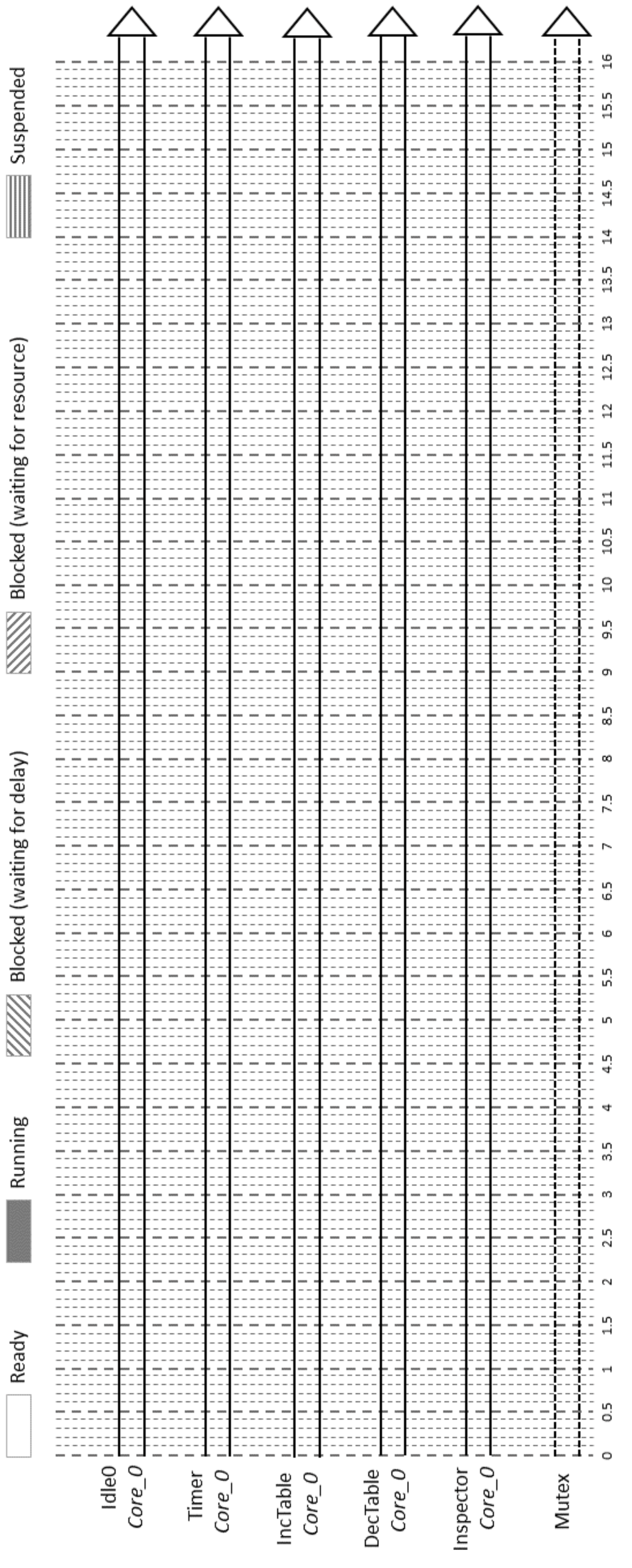


Figure 3.7 – *Program with Task inspector on the Core_0 and Mutex.*

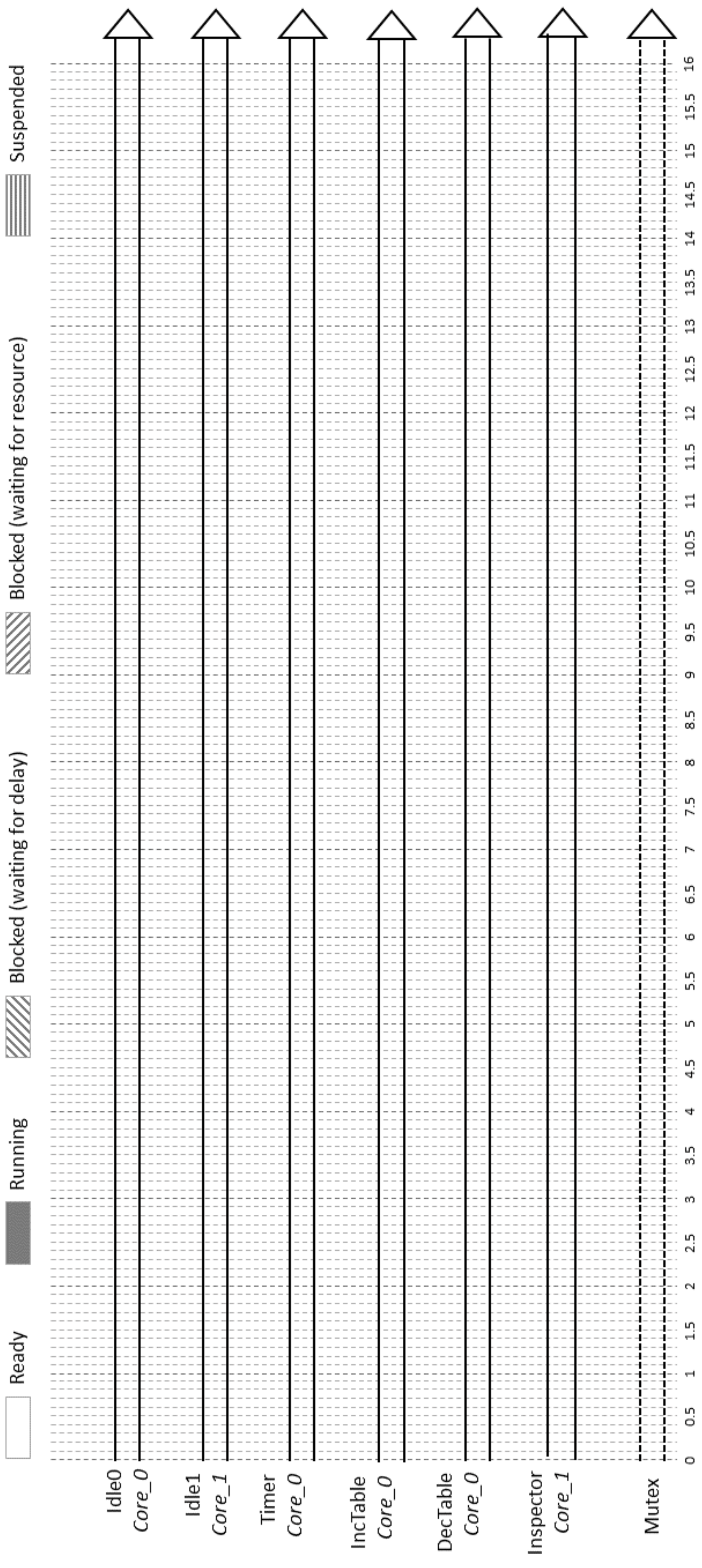


Figure 3.8 – *Program with Task inspector on the Core_1 and Mutex.*

LAB N° 4

Direct task notification

Lab Objectives

- Using task notification instead of semaphore.
- Using event group for a notification.

4.1 Direct task notification (Lab4-1)

A *direct to task notification* is an event sent directly to a task, rather than indirectly to a task via an intermediary object such as a queue, event group or semaphore. The *direct to task notification* is faster than using an intermediary objects and has a RAM Footprint benefits.

- Duplicate the « lab3-2_two_sem_clk » folder to « lab4-1_two_notifications_clk ».
- Answer the following questions :
  — Study the parameters of the *xTaskNotifyGive()* function. Web help

  — Study the parameters of the *ulTaskNotifyTake()* function. Web help

- Replace the two separate semaphores (i.e. notification of the *IncTable* and *DecTable* tasks) by a the *direct to task notification* mechanism. For the *ulTaskNotifyTake()* function, we set the first parameter *xClearCountOnExit* to *TRUE*. Print the return value (pending event counter) of *ulTaskNotifyTake()* function for each task.

- Run the program and explain the behavior.

- In the *Timer* task, add a new *give()* notification to *IncTable* task. So, you have 2 notifications in a row to the *IncTable* task.
- Run the program and explain the behavior.

- We set now the first parameter *xClearCountOnExit* to *FALSE* for the *ulTaskNotify-Take()* function. What is the difference of behavior regarding to the previous question ?

4.2 Direct task notification with a event value (Lab4-2)

To illustrate the principle, we are going to modify the algorithm of the *Timer* task. Its functional behavior is as follows :

```
Task Timer is
    Properties: Priority = 5, ACTION = eSetBits
    Out        : Clk is event
    Cycle :
        waitForPeriod(250 ms);
        computeTime(20 ms);
        print("Task Timer : Notify Give (count=%d)", count);
        // IncTable task notifications
        notify(incTableHandler, (0x01 « count), eSetBits);
        notify(incTableHandler, (0x02 « count), ACTION);
        // DecTable task notification
        notify(decTableHandler, (0x01 « count), eSetValueWithoutOverwrite);
        // counter modulo 4
        count = (count + 1)
    end
end
```

Algorithm 5: New Timer algorithm.

1. Duplicate the « lab4-1_two_notifications_clk » folder to « lab4-2_two_notifications_clk2 ».
2. Answer the following questions :
   - Study the parameters of the *xTaskNotify()* function. Web help
   - Study the parameters of the *xTaskNotifyWait()* function. Web help
3. Modify the code of *Timer* task.
   - The *ACTION* property can be a constant value and can take the *eSetBits*, *eSetValueWithoutOverwrite* or *eSetValueWithOverwrite* value.
   - We set by default the *ACTION* property to *eSetBits*.
   - What is the best value of the first parameter (*ulBitsToClearOnEntry*) of the *xTaskNotifyWait()* function ?

- We set the second parameter (*ulBitsToClearOnExit*) of the *xTaskNotifyWait()* function to 0.

4. Run the program, copy the console until 107 ticks and explain the behavior, in particular the pending counter value in each 2 tasks.

5. We set the *ACTION* property to *eSetValueWithoutOverwrite*. Run the program, copy the console until 107 ticks and explain the behavior.

6. We set the *ACTION* property to *eSetValueWithOverwrite*. Run the program, copy the console until 107 ticks and explain the behavior.

LAB N° 5

Application

Lab Objectives

- Use the knowledge previously seen on freertos services
- Using GPIO
- Using Interrupt

5.1 Specification of the application

We want to implement the functional structure depicted in figure 5.1.

5.1.1 Scan task

The task is activated on *ScanH* event. It acquires the value of the *Value* variable, compares it to high and low thresholds. *Value* being a theoretical output of a Digital/Analog converter not available for this Lab. It is possible to simulate its evolution by a random generation made by *ScanH* when activated.
The *rand()* function allows you to generate a number between 0 and RAND_MAX. So you just need, by rule of three, to bring the result between 0 and 100 and to set the low threshold at 15 and the threshold at 85.
If one of the thresholds is exceeded, a message of *High alarm* or *Low alarm* type is sent to the *Mess* queue, accompanied by the threshold value of *Value* variable. Otherwise, the value of *Value* variable is assigned to *SampleValue*.

```
static const uint32_t VALUE_MAX = 100;
static const uint32_t LOW_TRIG = 15;
static const uint32_t HIGH_TRIG = 85;
```

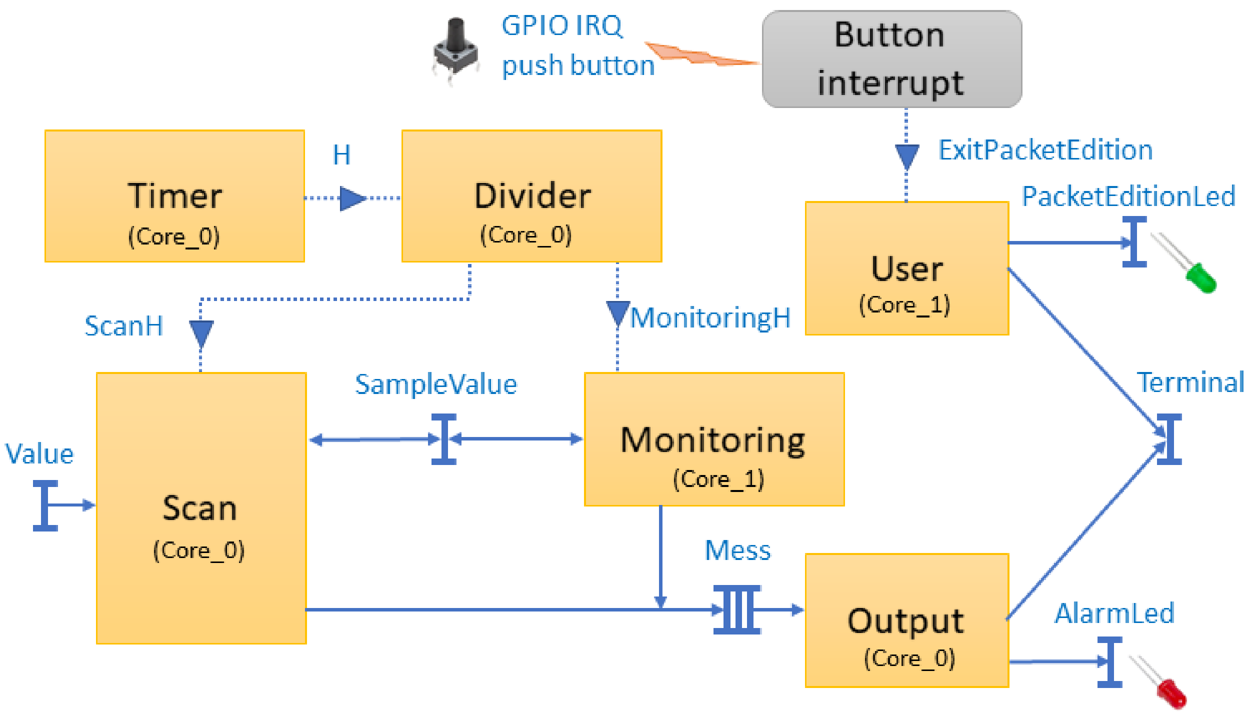


Figure 5.1 – *Functional description.*

```
int iValue = rand() / (RAND_MAX / VALUE_MAX);
if (iValue < LOW_TRIG) {
    // Low Alarm
}
else if (iValue > HIGH_TRIG) {
    // High Alarm
}
else {
    // No Alarm
}
```

5.1.2 Monitoring task

The task is activated on *MonitoringH* event. It sends messages of *Monitoring* type in the *Mess* queue. These messages contain the value of the variable *SampleValue*. Thus, the structure of *Mess* queue can be described as follows :

```
typedef enum typeInfo { Alarm, Monitoring };
typedef struct
{
  enum typeInfo xInfo;
  int xValue;
} typeMessage;
```

5.1.3 Timer task

The *Timer* task generates a periodic *H* event of 10ms.

5.1.4 Divider task

The *Divider* task performs a division of *H* event by 5 to generate *ScanH* event and a division by 18 to generate *MonitoringH* event.

5.1.5 User task

The task is activated by the keyboard. It allows the user to print a packet of characters on the terminal (called *Packet Edition*). These characters are edited by packet, the end of a packet corresponding to the character @ (for example, a packet is « `it is an example@` » as depicted Listing 5.1). When entering a character packet (*Packet Edition* mode), the *Output* function should not display its messages on the terminal and the *PacketEditionLed* is set to 1 (the green led is ON). After entering the character @ (end of *Packet Edition* mode), it can display as many messages as necessary, until the user enters the next character. The characters can be entered using the *getch()* function. An example of use is depicted below :

```
/* Scan keyboard every 50 ms */
while (car != '@') {
  car = getch();
  if (car != 0xff) {
    printf("%c", car);
    }
  vTaskDelay(pdMS_TO_TICKS(50));
}
```

Another solution for exiting *Packet Edition* mode is to press the push button that triggers an interrupt (named *Button interrupt*). This interrupt sends an event (semaphore) to the *User* task so that it can exit *Packet Edition* mode.

5.1.6 Output task

The task displays a formatted message of the *Mess* queue as depicted in the terminal in Listing 5.1. When the message is an alarm, the *AlarmLed* must be set to 1 (the red led is ON).

Listing 5.1 – *Console example of the application*

```
13:0> Monitoring: Value = 69
203:0>  Monitoring: Value = 55
393:0>  Monitoring: Value = 20
583:0>  Monitoring: Value = 84
613:0>  Alarm: Value = 6
733:0>  Alarm: Value = 87
773:0>  Monitoring: Value = 85
```

```
853:0>  Alarm: Value = 98
913:0>  Alarm: Value = 8
963:0>  Monitoring: Value = 15
973:0>  Alarm: Value = 90
1033:0> Alarm: Value = 10
1093:0> Alarm: Value = 92
1153:0> Monitoring: Value = 50
1213:0> Alarm: Value = 87
1343:0> Monitoring: Value = 69

User mode
hello !@
End User mode
2393:0> Alarm: Value = 3
2393:0> Monitoring: Value = 15
2393:0> Alarm: Value = 9
```

5.2 Implementation of the application (Lab5)

1. Create the « lab5_application » lab from « esp32-vscode-project-template » GitHub repository.
2. Copy the provided « my_helper_fct.h » file to the « main » folder.
3. Overwrite the « main.c » file by the provided code of the « lab5_main.c » file.
4. Create a « sdkconfig.defaults » file with right parameters.
5. Write the program for *Timer*, *Divider*, *Scan* and *Monitoring* tasks. The capacity of *Mess* queue is 5. The tasks are mapped on different cores referenced in the figure 5.1.
6. Build to check that is no compilation error.

In order to validate the behavior of the application and to clearly highlight the management of *Mess* queue, we can proceed as below :

1. Validate the functional structure with only the *Timer*, *Divider*, *Scan* and *Monitoring* tasks. Check that after a number of message submissions corresponding to the maximum capacity of the *Mess* queue, the *Monitoring* and *Scan* tasks are blocked.
2. Wire the 2 leds on the board.
3. Add the *Output* task. We will check that the *Monitoring* and *Scan* tasks are no longer blocked. Only the *Output* task can be blocked on an empty *Mess* queue. Check the behavior of the red led.
4. Add the *User* task without using interrupt. Note that the *getch()* function also uses the terminal as the *printf()* function. You must protect the simultaneous write on the terminal. Check the application and green led behaviors.
5. Wire the push button on the board.

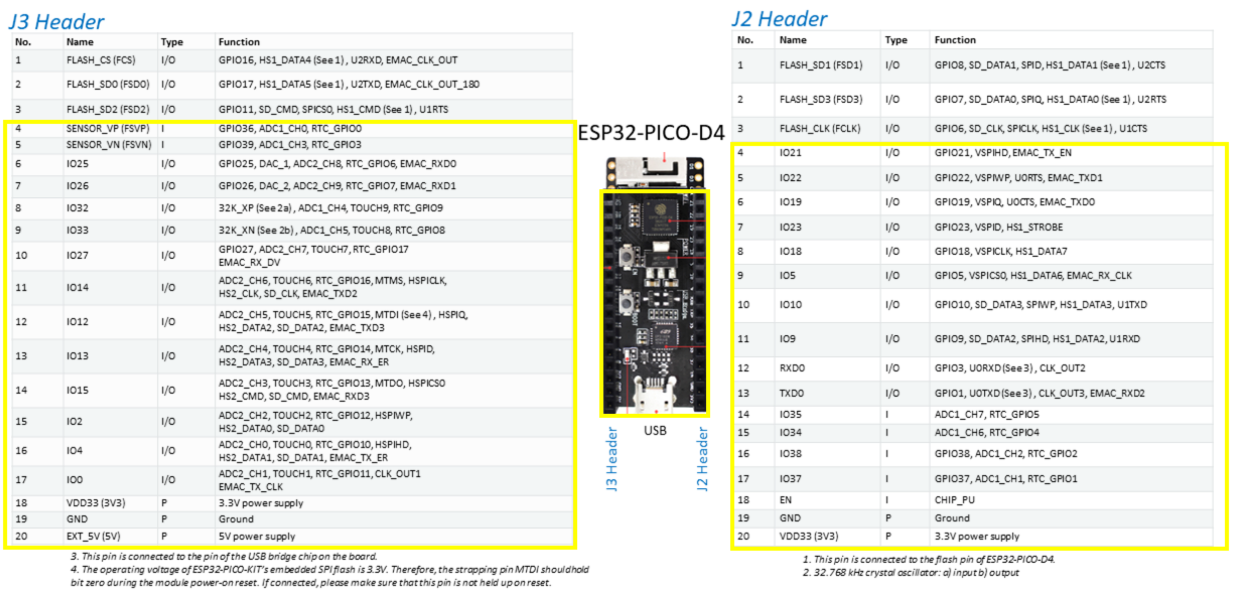
6. Add the *button interrupt* function and check the behavior. The name of the *give()* function is different when called from an interrupt (Web help).

Part IV

Appendix

ANNEXE A

ESP32 Board



J3 Header

No.	Name	Type	Function
1	FLASH_CS (FCS)	I/O	GPIO16, HS1_DATA4 (See 1) , U2RXD, EMAC_CLK_OUT
2	FLASH_SD0 (FSD0)	I/O	GPIO17, HS1_DATA5 (See 1) , U2TXD, EMAC_CLK_OUT_180
3	FLASH_SD2 (FSD2)	I/O	GPIO11, SD_CMD, SPICS0, HS1_CMD (See 1) , U1RTS
4	SENSOR_VP (FSVP)	I	GPIO36, ADC1_CH0, RTC_GPIO0
5	SENSOR_VN (FSVN)	I	GPIO39, ADC1_CH3, RTC_GPIO3
6	IO25	I/O	GPIO25, DAC_1, ADC2_CH8, RTC_GPIO6, EMAC_RXD0
7	IO26	I/O	GPIO26, DAC_2, ADC2_CH9, RTC_GPIO7, EMAC_RXD1
8	IO32	I/O	32K_XP (See 2a) , ADC1_CH4, TOUCH9, RTC_GPIO9
9	IO33	I/O	32K_XN (See 2b) , ADC1_CH5, TOUCH8, RTC_GPIO8
10	IO27	I/O	GPIO27, ADC2_CH7, TOUCH7, RTC_GPIO17 EMAC_RX_DV
11	IO14	I/O	ADC2_CH6, TOUCH6, RTC_GPIO16, MTMS, HSPICLK, HS2_CLK, SD_CLK, EMAC_TXD2
12	IO12	I/O	ADC2_CH5, TOUCH5, RTC_GPIO15, MTDI (See 4) , HSPIQ, HS2_DATA2, SD_DATA2, EMAC_TXD3
13	IO13	I/O	ADC2_CH4, TOUCH4, RTC_GPIO14, MTCK, HSPID, HS2_DATA3, SD_DATA3, EMAC_RX_ER
14	IO15	I/O	ADC2_CH3, TOUCH3, RTC_GPIO13, MTDO, HSPICS0 HS2_CMD, SD_CMD, EMAC_RXD3
15	IO2	I/O	ADC2_CH2, TOUCH2, RTC_GPIO12, HSPIWP, HS2_DATA0, SD_DATA0
16	IO4	I/O	ADC2_CH0, TOUCH0, RTC_GPIO10, HSPIHD, HS2_DATA1, SD_DATA1, EMAC_TX_ER
17	IO0	I/O	ADC2_CH1, TOUCH1, RTC_GPIO11, CLK_OUT1 EMAC_TX_CLK
18	VDD33 (3V3)	P	3.3V power supply
19	GND	P	Ground
20	EXT_5V (5V)	P	5V power supply

3. This pin is connected to the pin of the USB bridge chip on the board.
4. The operating voltage of ESP32-PICO-KIT's embedded SPI flash is 3.3V. Therefore, the strapping pin MTDI should hold bit zero during the module power-on reset. If connected, please make sure that this pin is not held up on reset.

J2 Header

No.	Name	Type	Function
1	FLASH_SD1 (FSD1)	I/O	GPIO8, SD_DATA1, SPID, HS1_DATA1 (See 1) , U2CTS
2	FLASH_SD3 (FSD3)	I/O	GPIO7, SD_DATA0, SPIQ, HS1_DATA0 (See 1) , U2RTS
3	FLASH_CLK (FCLK)	I/O	GPIO6, SD_CLK, SPICLK, HS1_CLK (See 1) , U1CTS
4	IO21	I/O	GPIO21, VSPIHD, EMAC_TX_EN
5	IO22	I/O	GPIO22, VSPIWP, U0RTS, EMAC_TXD1
6	IO19	I/O	GPIO19, VSPIQ, U0CTS, EMAC_TXD0
7	IO23	I/O	GPIO23, VSPID, HS1_STROBE
8	IO18	I/O	GPIO18, VSPICLK, HS1_DATA7
9	IO5	I/O	GPIO5, VSPICS0, HS1_DATA6, EMAC_RX_CLK
10	IO10	I/O	GPIO10, SD_DATA3, SPIWP, HS1_DATA3, U1TXD
11	IO9	I/O	GPIO9, SD_DATA2, SPIHD, HS1_DATA2, U1RXD
12	RXD0	I/O	GPIO3, U0RXD (See 3) , CLK_OUT2
13	TXD0	I/O	GPIO1, U0TXD (See 3) , CLK_OUT3, EMAC_RXD2
14	IO35	I	ADC1_CH7, RTC_GPIO5
15	IO34	I	ADC1_CH6, RTC_GPIO4
16	IO38	I	GPIO38, ADC1_CH2, RTC_GPIO2
17	IO37	I	GPIO37, ADC1_CH1, RTC_GPIO1
18	EN	I	CHIP_PU
19	GND	P	Ground
20	VDD33 (3V3)	P	3.3V power supply

1. This pin is connected to the flash pin of ESP32-PICO-D4.
2. 32.768 kHz crystal oscillator: a) input b) output

FIGURE A.1 – *Pin description of ESP32-PICO-D4 board.*

79

References